CODEN: LUTFD2/(TFRT-5347)/1–136/(1986)

SIP-SYSTEMET

Ett datorsystem för reglertekniska experiment och mätdatabehandling

Anders Birkedal

Institutionen för Reglerteknik
Lunds Tekniska Högskola
Februari 1986

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
February 1986

Document Number
CODEN: LUTFD2/(TFRT-5347)/1–136/(1986)

Author(s)
Anders Birkedal

Supervisor
Björn Wittenmark

Sponsoring organisation

Title and subtitle
SIP-systemet – Ett datorsystem för reglertekniska experiment och mätdatabehandling (The SIP-system – a computer system for experiments in automatic control and experimental data processing)

Abstract

The research program of TFL (Swedish Newsprint Research Centre) has created a need for developing software for a PDP-computer, connected with a process interface, with the following specifications:

- Automatic control of a pilot plant used for experiments
- Experiments with different kinds of automatic control
- Storing experimental data for processing in a VAX computer
- Graphical and numerical presentation on computer terminal display
- Simulation of experiments in a VAX computer
- Reconstruction of experiments in a VAX computer
- Easy to change for new applications and new computer equipment
- Easy to handle, safety against human faults

The created system contains a number of concurrent processes, and common routines. All software development is made in a VAX computer during simulation of the hardware environment. For creating new applications of the system, only one concurrent process needs to be changed (the control loop). Other information of different applications, like display design and connections to process interface, is written in data files which the system reads while starting up. The SIP-system contains routines for automatic creation of a new edition of the system, including every new application.

While running the system, a number of standard commands are defined, and the system ignores all wrong and unreasonable commands. While designing and testing the SIP-system, it has been used for direct digital control of a paper rewinder.

The SIP-system is written in Fortran, and the letters SIP stands for Styrning, Insamling, Presentation (Controlling, Collecting and Presentation).

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
Swedish

Number of pages
136

Recipient's notes

Security classification

SIP-Systemet

Ett datorsystem för reglertekniska experiment och mätdatabehandling

Anders Birkedal

INNEHÅLLSFÖRTECKNING

1 INLEDNING

Tidningspappersbrukens Forskningslaboratorium, i fortsättningen kallat TFL, bedriver forskning angående körbarhet hos pappersmaskiner, rullmaskiner och tryckpressar. Inom ramen för detta har projektet "Banspänningskontroll i rotationspressar" blivit utfört.

Målet för detta är att genom utjämning av variationer i banspänning, kunna minska banbrottsfrekvensen.
Detta sker bland annat genom studier av hur banspänningen uppför sig i en experimentmaskin, kallad LINDA, som konstruerats på TFL. En naturlig gren av detta projekt är att med reglertekniska experiment försöka hitta en metod att utjämna banspänningsvariationerna.

För att utföra bl.a. experiment av den typen, har TFL i sin ägo en dator av märket PDP, som kan kopplas till mät- och styrsignaler från försöksuppställningen.

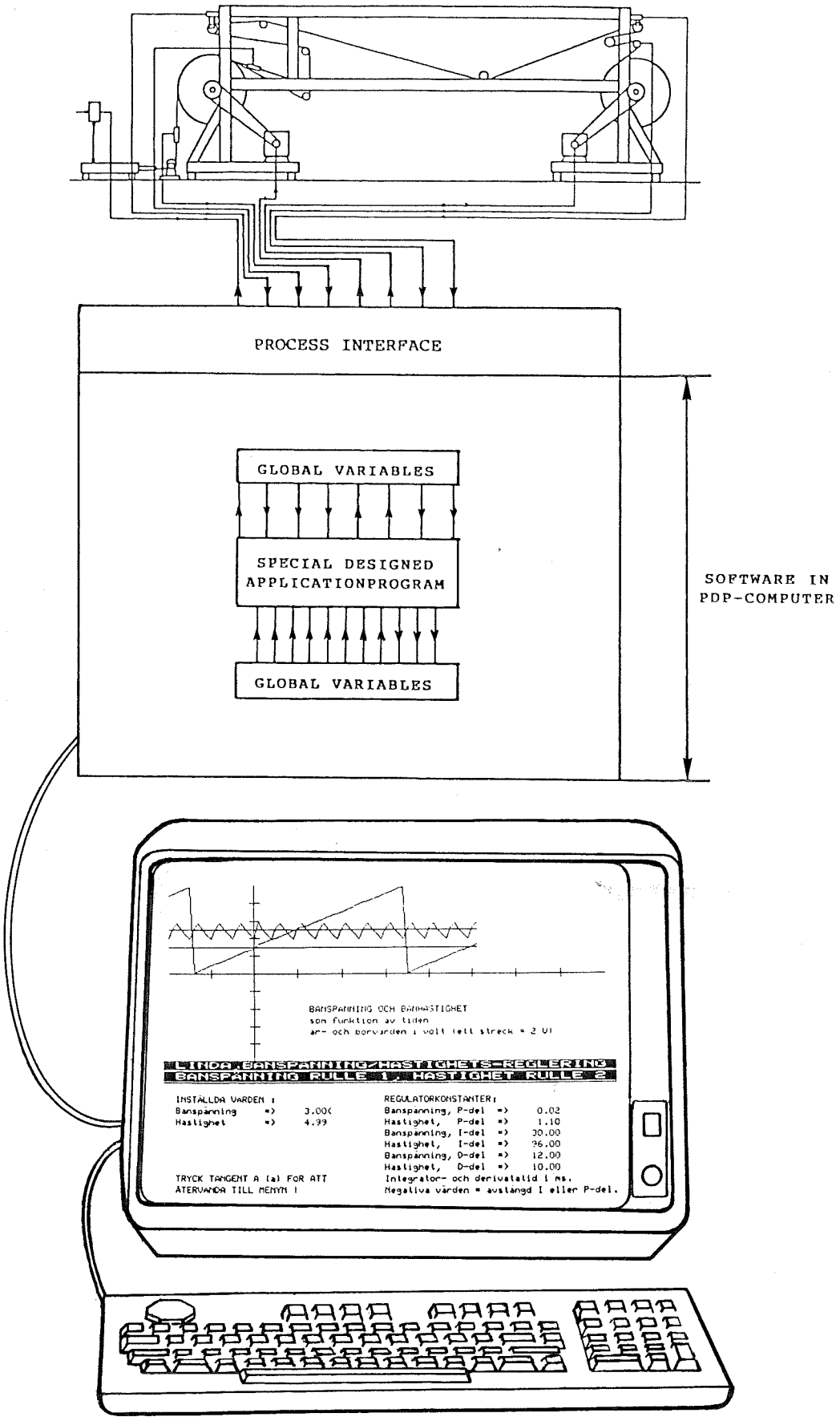
Detta har skapat ett behov av att utveckla mjukvara till denna dator, för att kunna utföra dessa experiment. Dessa önskemål kombinerade med TFL:s övriga önskemål om användning av PDP-datorn och övrig datorutrustning på företaget, har som resultat givit de kravspecifikationer på datorsystemet, vilka formuleras i kapitel 2, och kan sammanfattas i följande punkter:

Kravspecifikationer på SIP-systemet:

1. Styra försöksanläggning vid allmänna experiment
2. Speciellt kunna användas för reglertekniska experiment
3. Insamla mätdata och lagra dessa för senare analys i VAX-dator
4. Presentera data grafiskt och numeriskt på terminalen
5. Simulera laboratoriemiljö i VAX-dator
6. Utföra all programutveckling på VAX, så att programvaran är direkt användbar i PDP-datorn
7. Lätt att anpassa för olika tillämpningar och ny datorutrustning
8. Enkelt att handha, garderat mot mänskliga fel

I kapitel 3 diskuteras hur reglertekniska tillämpningar bör utföras i den givna miljön, med programspråket Realtids-Fortran som grund. Dessutom härleds en Fortranalgoritm för en PID-regulator.

I kapitel 4 utföres konstruktionen av mjukvaran så att den uppfyller kravspecifikationerna. Realtidsrutiner för VAX-Fortran skapas så att de får identiska anrop med motsvarande rutiner i PDP-datorns Realtidsfortran. Säkerheten mot felaktiga ingrepp från användarna, löses genom lättfattliga och klart definierade gränssnitt gentemot dessa. Figur 1.1 illustrerar hur operatören kan betrakta allt mellan försöksuppställningen och terminalen som en "black box",



Figur 1.1 SIP-systemet sett ur användarnas synvinkel, vid reglering av rullmaskin.

medan den som skall konstruera nya tillämpningar av datorsystemet, endast behöver skriva en programmodul (t.ex. en reglerloop), som kommunicerar med resten av systemet via globala variabler. Den "fasta" delen av systemet består av processmoduler och vanliga sekventiella program, som är lika för alla tillämpningar.

Kapitel 5 innebär en presentation av SIP-systemet, vilken utgör en sammanfattning av den utförligare dokumentationen som finns inbyggd i systemet på datorn. Alla standardmässiga filer och nya upplagor av dokumentationen automatgenereras med hjälp av Opiab-Konsults SMS (Software Management System), vilket presenteras i korta drag. Förkortningen SIP står för:
-STYRNING (direkt digital reglering)
-INSAMLING (av mätdata)
-PRESENTATION (av mät- och styrdata på grafisk terminal)

Vid körning av systemet, vilket beskrives i kapitel 6, utnyttjas ett antal standardiserade kommandon, och ignorerar felaktiga och orimliga kommandon.

En standardmall för utveckling av nya tillämpningar av systemet, vilket skall kunna utföras utan specialkunskaper i realtidsprogrammering, presenteras i kapitel 7. Förutom applikationsprogrammet (processmodul) enligt Figur 1.1, skall ett antal kommandofiler och indatafiler skrivas, vilka bland annat beskriver vilka av SIP-systemets processer som skall ingå i den aktuella varianten, dessa processers prioriteter och samplingstider, samt hur displayen skall se ut.

Ett exempel på hur SIP-systemet används för att reglera en rullmaskin (se Figur 1.1), ges i kapitel 8. I den redovisade varianten används PID-regulatorn från kapitel 3. Dessutom har möjligheten att välja mellan olika driftsfall via en meny på terminalskärmen införts.

SIP-systemet är utvecklat i TFL:s regi sommaren och hösten 1985 av Anders Birkedal i samarbete med Anders Jonsson, OPIAB-KONSULT. Författaren av denna rapport (Anders Birkedal) har använt detta arbete som examensarbete vid Lunds Tekniska Högskola. Härmed riktas ett tack till TFL och dess forskningschef Lars O Larsson för möjligheten att få utföra detta arbete samt till handledaren Leif Eriksson för det goda samarbetet i samband med arbetets utförande. Författaren vill även rikta ett tack till Anders Jonsson, som bidrog med sina fackkunskaper, Inger Björklund och Eva Petrov som hjälpte till med renskrivning, samt Fredrik Kjellander som renritade ett flertal av figurerna.

2 FÖRUTSÄTTNINGAR

2.1 Inledning

Inom ramen för TFL:s forskningsverksamhet i projektet "Banspänningskontroll i rotationstryckpressar" har framkommit ett behov av att utveckla ett system för digital kontroll och datoriserad mätdatabehandling.

Förutsättningen har varit att med befintlig utrustning (med eventuella smärre modifieringar) utveckla mjukvara som uppfyller de önskemål som kan tänkas uppstå inom TFL:s forskarlag vid detta och andra liknande projekt.

Första momentet av detta utvecklingsarbete är då att analysera vilka användningsområden som kan bli aktuella för detta system, och med hänsyn tagen till de begränsningar som finnes i och med den befintliga utrustningen, formulera vilka prestanda som systemet skall kunna uppfylla.

Programutvecklingen förutsättes börja "från början" med befintliga operativsystem och kompilatorer som grund.

2.2 Befintlig datorutrustning

Miljön för det datorsystem som skall utvecklas är skisserad i Figur 2.1. Kontakten med försöksuppställningen sker via ett processinterface (RTP), som består av ett antal olika (krets-) kort som fyller funktionerna digital ingång, digital utgång, analog ingång med A/D-omvandlare, analog ingång med A/D-om-

vandlare och anti-aliasingfilter samt analog utgång med D/A-omvandlare.

De olika korten kommunicerar direkt med datorn via en buss, så att man via adressering erhåller, eller styr ut digitala värden direkt med mjukvaran i datorn.

Följande tillgängliga in- och utgångar finns i systemet:

Beteckning på kortet och anslutningarna	Input/ Output	Digital/ Analog	Beskrivning
7436/43	I	A	16 singel eller 8 diff-ingångar Ej filter
7436/44	I	A	8 diff-ingångar Anti-aliasingfilter (single pole) Brytfrekvens 11 Hz
7438/20	I+O	D	16-bitar in och 16-bitar ut
7435/82	I	D	16-bitar in
7437/37	I	D	16-bitar, optiskt isolerad
7435/81	O	D	16-bitar, optiskt isolerad
7455/31	O	A	4 utgångar
7455/22	O	A	1 utgång

A/D- och D/A-omvandlarna har 11 bitars upplösning (12 om man räknar teckenbiten), dvs. de klarar av talområdet +2047 till -2048. Detta skall motsvara ett spänningsområde på de ingående analoga signalerna på +10.00-10.24 V till -10.00-10.24 V. Empiriska mätningar ger värdet +10.21 till -10.21. Omräkningsfaktorer för omvandlingen framgår av exemplet i kap. 8. För de analoga utgångarna gäller halva värdet, både riktvärde enligt manual, och empiriskt uppmätt värde.

De digitala in/utgångarna är kopplade så att låg nivå är logisk etta, minst signifikant bit är markerad nr 16.

RTP-interfacet är kopplat till en PDP 11-dator med en grafisk TEKTRONIX 4105-terminal med tillhörande hardcopy-skrivare (TEKTRONIX 4695).

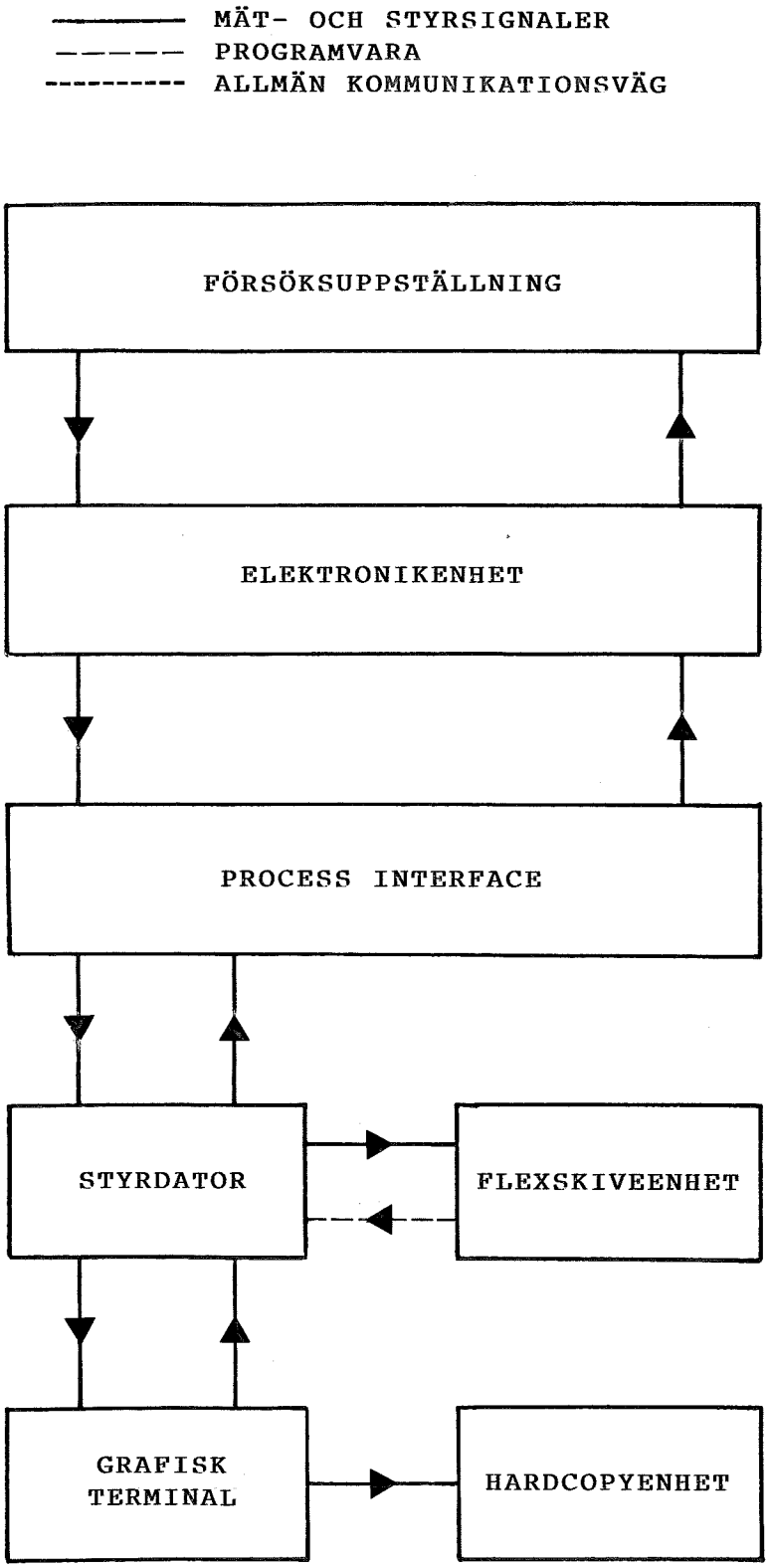
PDP-datorn har en systemklocka som går med nätfrekvensen 50 Hz. Experiment har visat att den snabbaste praktiskt genomförbara samplingsfrekvensen med SIP-systemet i PDP-miljön blir halva denna frekvens (25 Hz. periodtid 40 ms), vilken alltså blir SIP-systemets Klockfrekvens (se kap. 4.1).

Programutveckling, datalagring och simulering skall ske på en VAX-11/730-dator med tillgång till en terminal av samma typ som på PDP-datorn, se Figur 2.2.

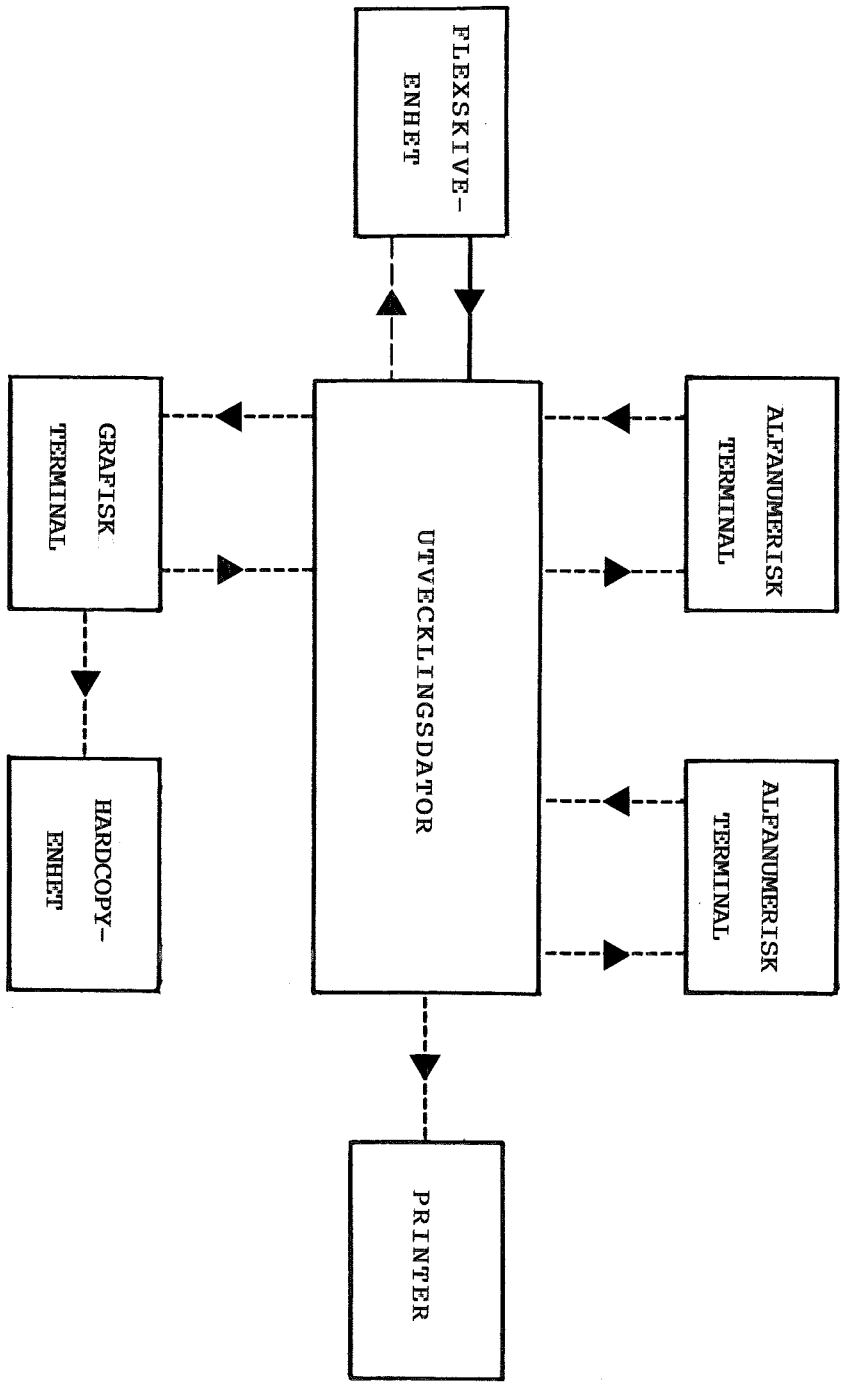
För bägge datorerna gäller att de är utrustade med FORTRAN-kompilatorer, men ej någon kompilator för något annat högnivåspråk.

Överföring av programvara och data mellan datorerna sker med hjälp av floppy-disk.

För utförligare information hänvisas till manualerna för RTP-interface, PDP-11, VAX-11 och Tektronix 4105 [8,10,11, 12,13,14,15,16].



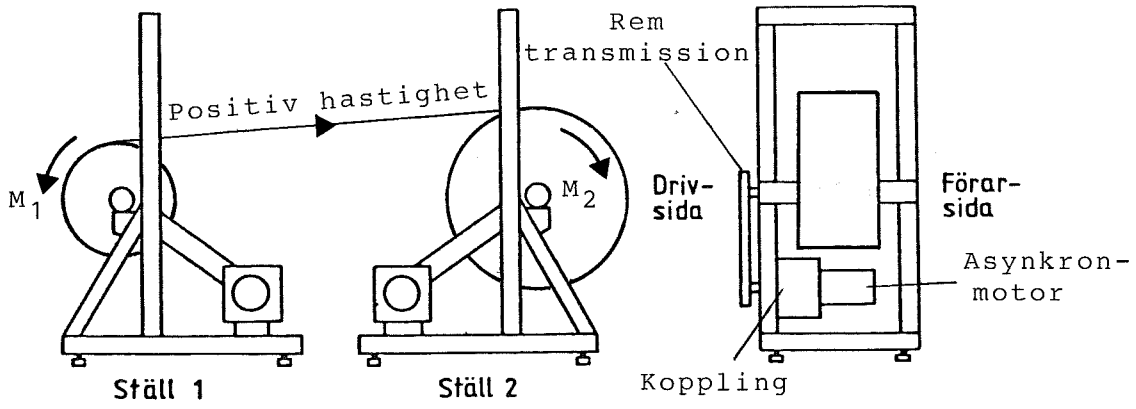
Figur 2.1 Schematisk bild av hårdvaran i laboratoriemiljön.



Figur 2.2 Schematisk bild av hårdvaran i miljö för utveckling och simulering.

2.3 Befintlig försöksuppställning

Pappersbanors uppförande i rullmaskiner och tryckpressar studeras på TFL genom försök i en maskin, kallad LINDA, som i grundutförande består av två rullställ enligt Figur 2.3.



Figur 2.3 Lindas rullställ.

Pappersrullarna drives av var sin asynkronmotor via en virvelströmskoppling. Momentet som påverkar rullen är proportionellt mot styrsignalen till kopplingen. Momentriktningen framgår av figuren. Momentsignalerna används för reglering av papperets hastighet och banspänning. Beroende på vilket moment som är störst blir hastigheten riktad i positiv (rulle 1 ⟶ rulle 2) eller negativ riktning. I regel används det ena momentet för reglering av banspänningen och det andra för reglering av hastigheten.

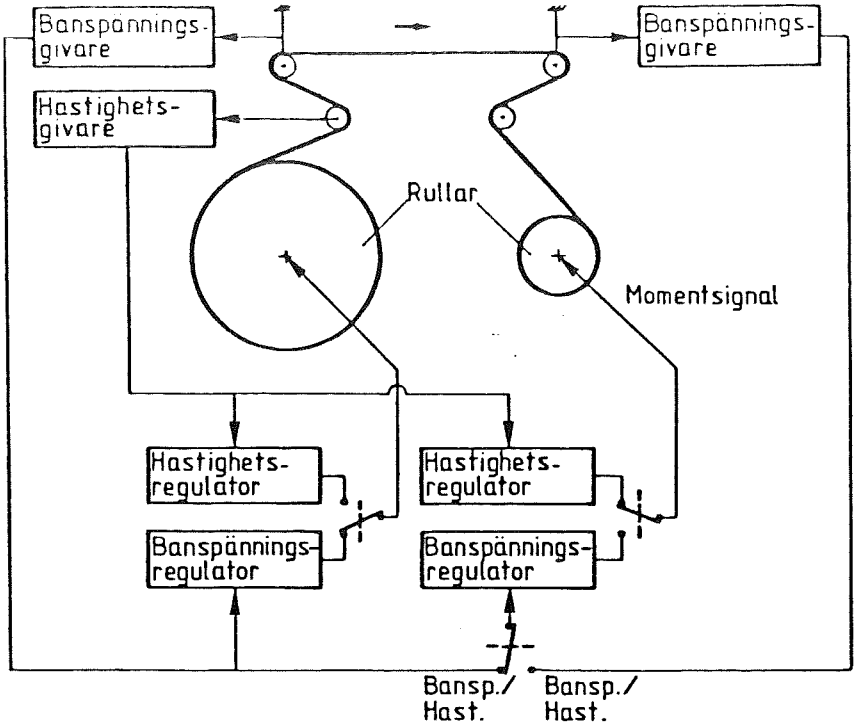
Till rullställen hör en styrenhet baserad på analoga elektronikmoduler.

Denna styrenhet sänder ut momentsignaler, och inhämtar följande mätdata:

Data	Erhållen signal
Banhastighet	-10 V - +10 V
Banspänning vid rullställ 1	-10 V - +10 V
-"- 2	-10 V - +10 V
Vinkelhastighet rulle 1	Pulser ~ varvtalet
-"- 2	Pulser ~ varvtalet
-"- 1	0 - 10 V
-"- 2	0 - 10 V
Banbrottsindikering	0 eller -15 V

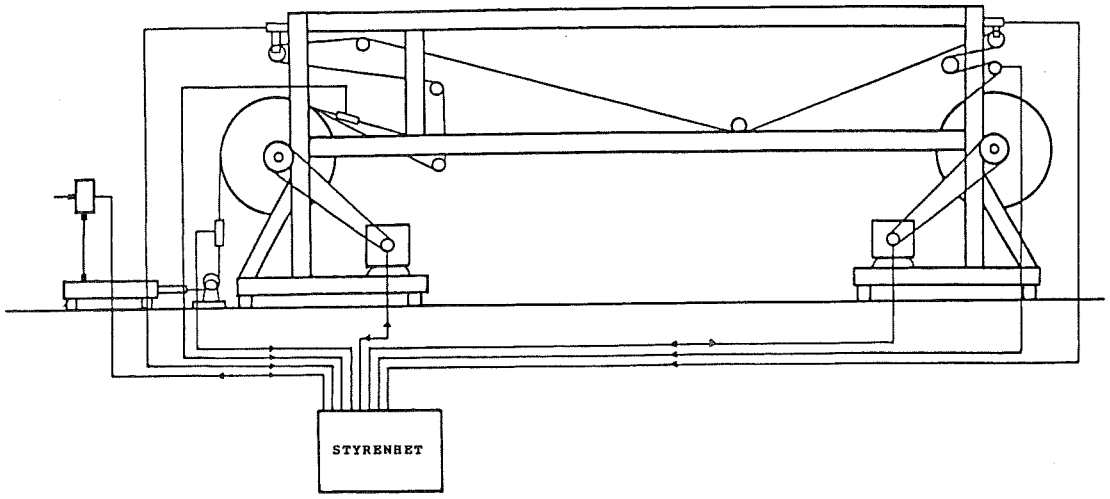
Mätsignaler, börvärden och styrsignaler, kan erhållas från utgångar på styrenheten. På denna enhet finns även ingångar för externa styrsignaler, vilka kan användas för direkt digital reglering med hjälp av PDP-datorn och RTP-interfacet.

Förutom externa momentsignaler, kan även externa börvärden till styrenhetens inbyggda PID-regulatorer, se Figur 2.4, ges. Omkoppling mellan olika alternativa styrsätt sker med digitala signaler.



Figur 2.4 Befintligt reglersystem med analoga PID-regulatorer.

I denna anläggning kan man simulera olika industriprocesser i laboratoriemiljö. Figur 2.5 visar ett exempel på försöks-uppställning för rullbromssystem i tryckpressar.



Figur 2.5 Exempel på en försöksuppställning,
försök med bromsband på rulle 1.

För utförligare beskrivning hänvisas till manualen för LINDA.

Referenser:

Wallin: Manualer för LINDA. [6,7]

Eriksson: Försöksanläggning för studier av processer med snabblöpande pappersbanor. [2]

2.4 Realtidsfortran

En förutsättning för detta arbete var att det skulle utföras i programspråket FORTRAN. FORTRAN är ett språk som i grunden ej är skapat för realtidstillämpningar. Det är ej heller lämpat för strukturerad programmering, vilket eftersträvas i detta arbete. Dock är det ett av de vanligast förekommande programspråken i industriella tillämpningar. Detta gäller även tillämpningar av realtidskaraktär. Motivet att använda FORTRAN i detta arbete är att det är standarspråk för TFL, och övergång till andra språk skulle medföra förutom investeringskostnaderna (kompilator och annan nödvändig mjukvara), ett behov av att omskola personalen.

Färdiga funktioner för realtidstillämpningar finns i PDP-datorns variant av FORTRAN. I denna "programdialekt" kan olika självständiga program köras som parallella processer (i detta fall benämns parallella processer med uttrycket tasks) genom att man först ger kommandot:

```
INSTALL PROGRAMNAMN
```

Då installeras programmet som en task, vilket innebär att när man gör:

```
RUN PROGRAMNAMN
```

så körs programmet som en självständig process i datorn, parallell med den process som är användarens körning på datorn. De olika parallella processerna kan kommunicera med varandra genom att skriva och läsa i gemensamma variabler vilka är samlade i COMMON-block. Dessa COMMON-block installeras precis som processerna i form av en BLOCK-DATA-modul enligt följande:

```
BLOCK DATA BLOCKNAMN
          :
          :
          :
DEKLARATIONER AV GEMENSAMMA VARIABLER
          :
          :
          :
END
```

För att synkronisera processerna (tasks) i förhållande till varandra och i förhållande till tiden, används EVENTFLAGS, vilka är heltalsvariabler som kan få värdet 0 eller 1.

Ett antal olika systemprocedurer finns tillgängliga för att reglera samspelet mellan de olika "tasks" man använder.

Exempel på sådana är:

```
CLREF(eventflag,status)
nollställ eventflag

SETEF(eventflag,status)
sätt eventflag (=1)

DECLAR(status)
ange att signifikant händelse har inträffat

WAITFR(eventflag,status)
vänta tills flagga sätts

WFLOR(eventflag1,eventflag2,.....eventflagn)
vänta på att någon av flaggorna sätts

MARK(eventflag,time,timeunit,status)
ange att flaggan skall sättas efter en viss tid
```

Referenser:

Elmqvist m.fl.: Datorer i reglersystem.
Realtidsprogrammering. [3]

Manualer till PDP 11. [14]

2.5 Målsättning

Den befintliga försöksuppställningen är försedd med ett system för reglering, och mätdatabehandling, baserad på analog elektronik. En av huvudmålsättningarna blir att det nya digitala systemet skall innefatta alla funktioner som redan finns i det gamla analoga systemet, för att bli oberoende av detta.

Systemet skall naturligtvis innebära en förbättring av funktionerna med hjälp av de fördelar som datorer ger. Rimliga önskemål på förbättringar:

* Presentationen av alla fakta som användaren önskar ha, skall kunna avläsas på dataskärmen.
* Datorns möjlighet till att rita grafiska presentationer av förlopp skall utnyttjas.
* Databehandling av mätdata skall underlättas genom att man skall kunna lagra mätdata och direkt läsa in det från databehandlingsprogrammet i VAX-datorn. Till skillnad från tidigare system då mätdata fick matas in för hand vid terminal.
* Regleringen av processen skall förbättras.
* Experiment med olika reglerstrategier skall kunna utföras, vilket ej har låtit sig göras med de fasta regulatorerna.

För att kunna utnyttja VAX-datorns kapacitet vid utveckling av nya tillämpningar av detta system, samt vid analys av ett mätdataförlopp vore det önskvärt med möjligheten att simulera laboratoriemiljön på VAX-datorn.

Systemet skall förutom att styra själva rullmaskinen, även kunna styra och mäta på olika försöksuppställningar som kan tänkas monteras på denna, t.ex. reglerbara valsar.

Ett programmeringsarbete av den omfattning som blir aktuell i detta fall, blir olönsamt om man gör ett alltför specialiserat program, varför det är önskvärt att utforma det så att så små modifieringar som möjligt behövs för att anpassa systemet till mera godtyckliga tillämpningar.

En annan aspekt är systemets snitt gentemot användarna. Det bör vara så enkelt som möjligt, och inte kräva mer specialistkunskaper än vad en "vanlig" forskare eller ingenjör kan tänkas ha.

Sammanfattningsvis kan målsättningen formuleras i följande punkter:

1. Styra försöksanläggningen tillfredsställande vid allmänna experiment.
2. Utföra reglertekniska experiment.
3. Insamla mätdata för analys i VAX-dator.
4. Presentera data på ett användarvänligt sätt.
5. Simulera laborationsmiljön i VAX-datorn.
6. Programutveckling skall kunna ske på VAX-datorn.
7. Lätt att anpassa för allmänna uppgifter.
8. Enkelt att handha, självinstruerande.

2.6 Avgränsning av problemet och kravspecifikationer

I förra kapitlet formulerades en allmän målsättning med datorsystemet. Här skall en mera detaljerad kravspecifikation formuleras med utgångspunkt från målsättningen och vad som är möjligt och rimligt med hänsyn till befintlig utrustning. En generell förutsättning är att systemet måste arbeta i realtid, med flera uppgifter som tar olika lång tid och behöver utföras parallellt.

Standardisering av kommandon

VAX-datorn som har betydligt högre kapacitet än PDP-datorn kan utnyttjas för simulering av laboratoriemiljön. Detta sker lämpligen genom att simulera en PDP-körning vid en körning på VAX. Detta förutsätter då att man skall kunna utnyttja PDP-datorns kommandon på VAX-datorn.

För att kunna göra programutvecklingen på VAX-datorn krävs att man håller sig strikt till standardfortran så att filerna direkt kan överföras mellan datorerna då tillverkarens "extrafinesser" inte finns på båda datorerna eller ser olika ut.

Standarden kan i vissa fall kringgås genom att man skapar ett bibliotek av rutiner där rutinernas programkod är olika på datorerna, men med identiska anrop. Exempel på detta är t.ex. vissa realtidsfunktioner, se kap. 2.4.

Lagring av data

Lagring av alla åtkomliga mätdata för senare användning skall kunna utföras. Detta skall kunna ske parallellt med och oberoende av eventuellt pågående reglering. Kommandon för att styra denna funktion skall kunna ges från terminalen utan att avbryta pågående presentation av aktuella händelser (typ. varningssignaler), så att operatören hela tiden kan se på skärmen vad som lagras.

Reglering

Indata skall inte ändras hos reglerprogrammet förrän kommando om detta ges, så att samstämmiga indata alltid samverkar och felaktiga indata undvikes. Önskvärd är möjligheten att editera indata på datorskärmen, samtidigt som utdata presenteras på denna. Reglerprogrammet får inte avbrytas eller försenas av andra aktiviteter under körningens gång. Samplingsintervallen måste vara tillräckligt korta för att klara av de snabbaste frekvenser, man önskar reglera,(p.g.a. Samplingsteoremet[4]) vilka kan erhållas genom en frekvensanalys av mätsignalerna.

PDP-datorns begränsningar

I praktiken bestämmer PDP-datorns kapacitet samplingstiderna samt begränsar möjligheten att använda grafiska presentationer.

Försök har visat att samplingstiden måste vara 2 tick = 40 ms eller längre. För vissa tillämpningar t.ex. operatörskommunikation måste samplingstiden i praktiken vara mycket längre än 40 ms. Detta medför ett krav på att samplingstiden skall kunna justeras individuellt för de olika uppgifterna som datorn skall lösa, samt att samplingstiden skall vara lätt att ändra.

Vid arbetets början fanns ett programpaket i datorerna för grafik (PLOT-10). Detta är av det omfånget att det inte får plats i PDP:ns primärminnesutrymme samtidigt med SIP-systemet. I stället plockas de nödvändigaste rutinerna ur grafikpaketet och får ingå i SIP-systemet.

Grafiken är arbetsam för PDP-datorn att producera (speciellt som omställning mellan grafisk och numerisk mod på terminalen krävs vid varje samplingstillfälle), varför användning av denna måste begränsas så att tid blir över för andra uppgifter.

Färggrafik är en ännu mera tidskrävande procedur, varför de delar av presentationen som uppdateras varje samplingstillfälle (numeriska och grafiska data) får presenteras utan färg.

Vid design av de delar av presentationsdisplayen som inte uppdateras så ofta (Ledtexter m.m.) bör man utnyttja möjligheten till färggrafik för att erhålla en mera ergonomiskt tilltalande arbetsmiljö.

Moduluppbyggnad

Två av målsättningarna var att systemet skall kunna användas vid reglertekniska experiment, samt att det skall vara lätt att anpassa till allmänna uppgifter.

Ett system avsett för enbart reglertekniska experiment kan lätt uppfylla det kravet genom att man kan ge regulatorstrukturer och regleralgoritmer i form av parametrar som bestäms av användaren.
Detta system skall dock vara mera allmänt uppbyggt, varför sådana finesser tar för mycket utrymme.

Enklare är då att göra ett moduluppbyggt system där enskilda moduler t.ex. en reglerloop lätt kan bytas ut utan att påverka resten av systemet. Ett lämpligt krav är då att så många av modulerna som möjligt skall ha så allmänna uppgifter att de inte skall behöva ändras under systemets livslängd.

De moduler som skall ändras vid olika tillämpningar skall så långt som möjligt vara standardiserade i sin utformning, vilket leder till tidsbesparingar och mindre felkällor.
Detta är också ett led i att göra systemet så användarvänligt som möjligt.

Användarna

De flesta användarna av SIP-systemet skall ej behöva special-kunskaper. För att kunna ta fram ett lämpligt gränssnitt gentemot dessa, är det nödvändigt att definiera vilka användarna är. Man kan definiera tre användarkategorier, med olika kunskaper, operatörer, konstruktörer och programmerare.

Operatören skall utnyttja ett färdigt system för experiment, typ reglering av processen.

Konstruktören skall konstruera nya tillämpningar av SIP-systemet åt operatören. (Operatör och konstruktör kan naturligtvis vara samma person).

Programmeraren är den person som kan gå in och modifiera själva systmet, dvs de moduler som är fasta sett ur konstruktörens synvinkel.

Systemets krav på operatören: Inga kunskaper i datateknik, men kunskap om experimentuppställningen som datorn är kopplad till.

Operatörens krav på systemet: Självinstruerande, säkerhet mot felaktiga ingrepp från operatören, t.ex. tryckning på fel tangent, orimlig indata m.m.

Systemets krav på konstruktören: Behärska aktuell tillämpning. Programmeringskunnande motsvarande inledande programmeringskurs vid teknisk högskola.

Konstruktörens krav på systemet: Ett minimum av utvecklings- och programmeringsarbete skall behöva göras, realtidsfunktionerna skall vara låsta i systemet, så att vanlig sekventiell programmering kan tillämpas enligt standardkoncept för systemets moduler.

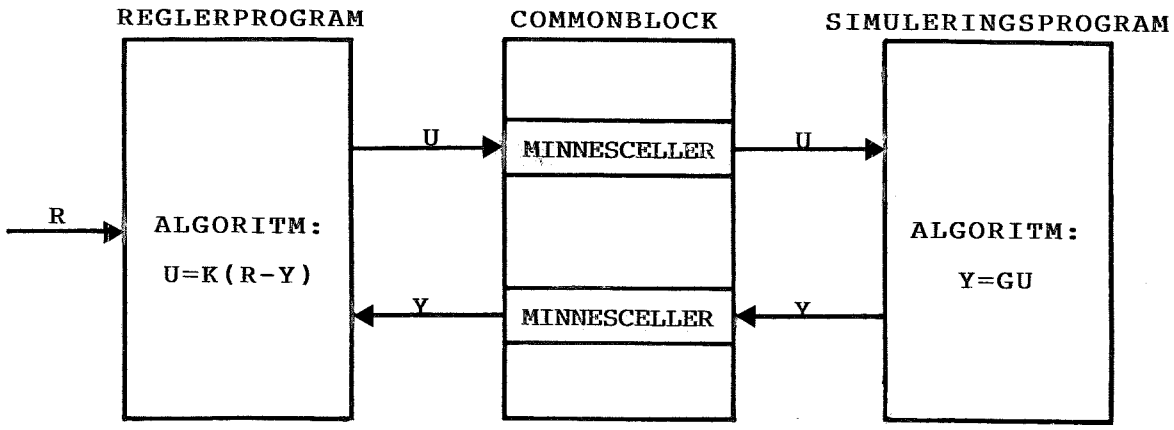
Systemets krav på programmeraren: Kunskaper i realtidsprogrammering och om de problem som kan uppstå i samband med realtidssystem. Att känna till något om datorernas operativsystem, och de praktiska tillämpnignarna av SIP-systemet är en fördel.

Programmerarens krav på systemet: Plats skall finnas för utbyggnad av systemet (med fler moduler t.ex.) utan att taga bort något som redan finns. Även de moduler som ej är avsedda att ändras, skall vara uppbyggda enligt samma standard som de övriga (och försedda med nödvändiga kommentarer).

3 REGLERTEKNISK PROBLEMANALYS

3.1 Simulering

Vid processer, som är dyra och komplicerade att göra experiment på, t.ex. pappersmaskiner, är det lämpligt att utnyttja en simulerad modell av processen för bestämning av regulatorns parametrar (finjustering måste naturligtvis ske experimentellt). I ett realtidssystem av den typ som eftersträvas här, kan detta ske enligt Figur 3.1.



Figur 3.1 Princip för simulering i ett moduluppbyggt Fortransystem.

I det tilltänkta SIP-systemet måste alltså ingå (minst) en modul för simulering som är utbytbar mot den (de) modul(er) som kommunicerar med den verkliga processen.
Observera att R, Y och U i Figur 3.1 kan vara vektorer, d.v.s. flera variabler.

Vid reglering av TFL:s egna experimentanläggningar, av samma typ som processen LINDA, är det mer tidsbesparande, och

därigenom mera lönsamt att utföra direkta experiment, då anläggningen i sig är byggd för att (mekaniskt) simulera industriprocesser, och därigenom är tillgänglig för full-skaleförsök för den som utför utvecklingsarbete på den.

Det krävs dock någon form av testprogram för att kunna utveckla resten av SIP-systemet i VAX-datorn (t.ex. regleringsmodulen), detta utföres lämpligen med hjälp av en likadan simuleringsmodul (d.v.s. ur systemets synvinkel exakt samma) som vid simulering av den verkliga processen, där processens överföringsfunktion ersätts med algoritmer som utför tester så att resten av systemet reagerar för olika signaler på ett riktigt sätt. Lämpliga sådana algoritmer kan t.ex vara generering av steg, ramper eller sinusvågor. I princip skulle den verkliga processens överföringsfunktion vara en mycket lämplig testalgoritm för ett program som skall reglera denna process, men då den från början är okänd, innebär det onödigt arbete att bestämma dess utseende i detta fall.

3.2 Variabel processdynamik

I vissa fall är de optimala reglerparametrarna för den valda styrlagen variabler, d.v.s. att när man genom simulering eller experiment ställer in vissa värden, så gäller dessa endast vid ett speciellt driftsfall.

Detta i sin tur beror på att processens dynamik, d.v.s. dess överföringsfunktion är variabel i tiden. Detta måste alltså medtagas vid bestämning av överföringsfunktionen G,

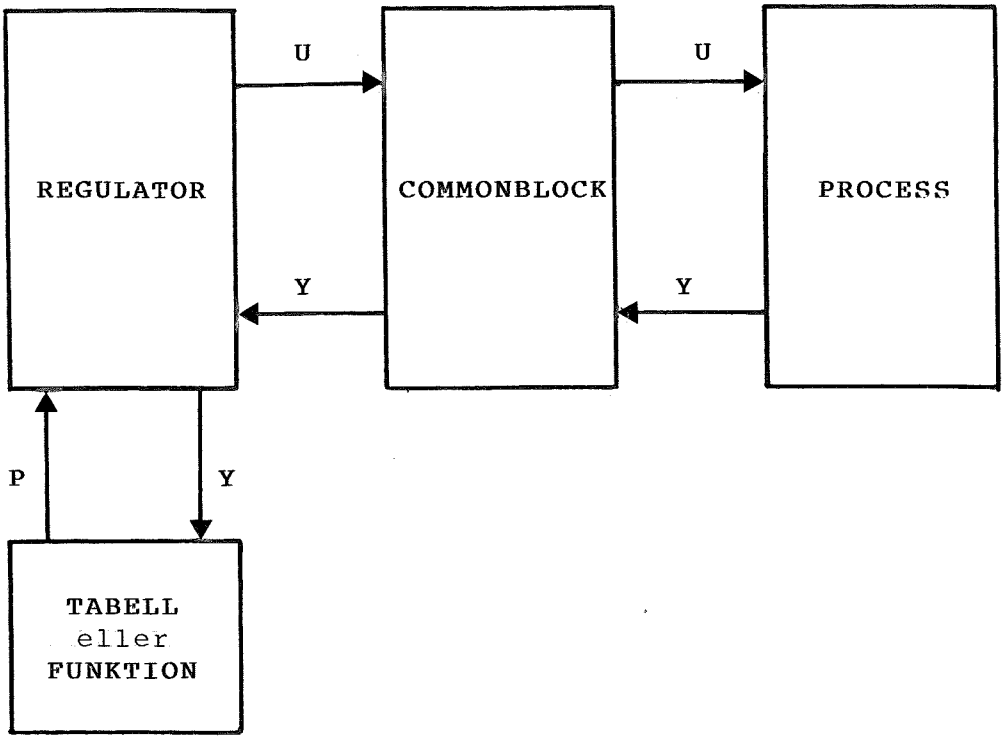
se Figur 3.1, vilket i sin tur medför en tidsvariabel regleralgoritm.

Anmärkning: Det som här omnämns som tidsberoende är egentligen ett beroende av driftsfall (som är tidsvariabla). En "vanlig" regleralgoritm har ju också ett tidsberoende och tiden ingår som variabel i de matematiska sambanden, medan funktionerna som beskriver dessa samband är konstanta i tiden.

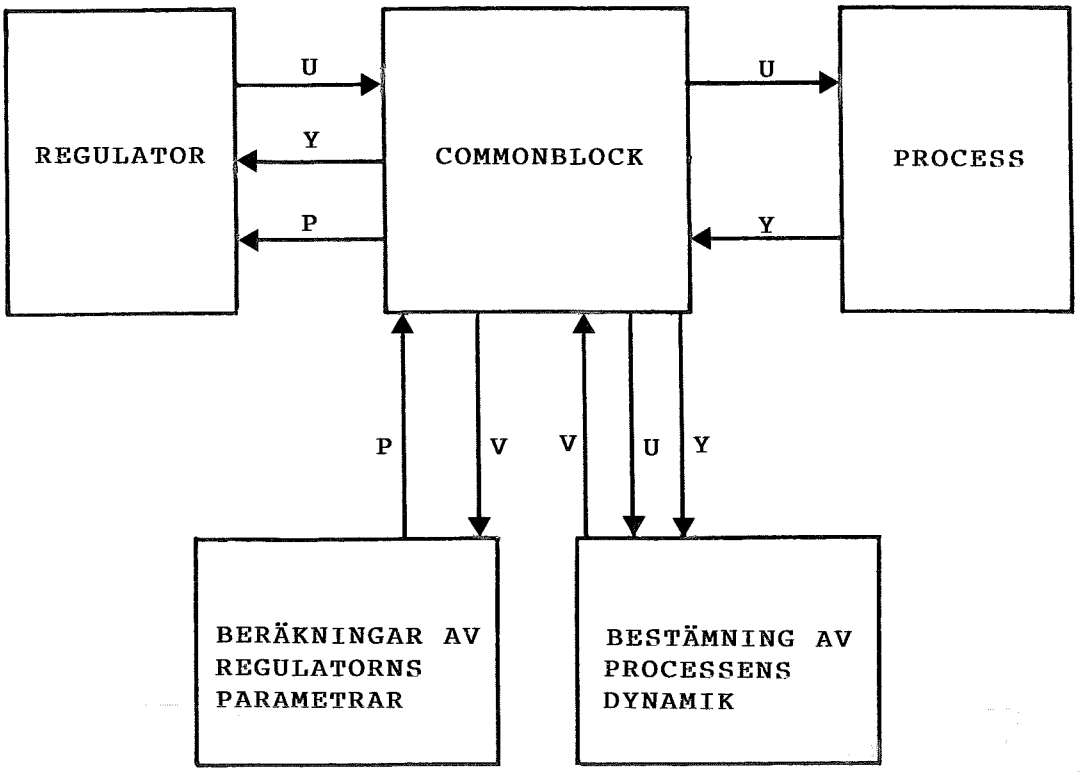
Ovanstående korrigering av regleralgoritmen kan utföras på två sätt:

1. Parameterstyrning: Genom experiment, simulering eller teoretisk analys bestäms parametrarnas tidsberoende i form av funktioner eller värdetabeller, se Figur 3.2.

2. Adaptiv reglering: Genom att införa en algoritm som kontinuerligt med hjälp av mätdata och styrsignaler bestämmer systemets överföringsfunktion för ögonblicket och räknar ut gällande parametervärden, erhålles en självinställande regulator, se Figur 3.3.



Figur 3.2 Parameterstyrning i moduluppbyggt Fortransystem.



V = Parametrar i processens överföringsfunktion

Figur 3.3 Adaptiv reglering i moduluppbyggt Fortransystem.

I fallet LINDA och liknande processer är det lämpligt att välja parameterstyrning, då adaptiv reglering leder till avancerad teori med olinjära system, och en parameterstyrningstabell direkt kan framställas genom experiment. Eventuell funktion för beräkning av reglerparametrar kan erhållas ur tabellen med numeriska metoder.

Detta innebär ett krav på att man skall kunna spara alla värden i ett common-block vid en viss tidpunkt, för att på så sätt framställa en tabell över sammanhängande värden för olika tidpunkter (d.v.s. driftsfall).

Detta krav sammanfaller med kravet på att lagra mätdata (se kap. 2.6) och kan lösas på samma sätt. Några speciella moduler för parameterskattning (Adaptiv reglering) kommer därför ej att ingå i SIP-systemets grundversion, såsom den redovisas i denna rapport, men den händige programmeraren skall inom ramen för SIP-systemet, kunna utöka antalet moduler. Att detta skall vara förberett ingår också i kraven på systemet.

3.3 PID-regulator

En digital PID-regulator kan härledas på följande sätt: Utgå från en beskrivning av styrlagen för en analog PID-regulator: [5]

$$U(t) = K\left[E(t) + \frac{1}{T_i}\int_{t_o}^{t} E(s)ds - T_d\frac{dY(t)}{dt} \right]$$

där

U = styrsignal

Y = mätsignal

E = R-Y = felet

R = referensvärde = börvärde

K = proportionell dimensionslös konstant

T_i = integrator-tidskonstant

T_d = derivata-tidskonstant

Derivatadelen opererar ej på felet (R-Y) utan direkt på Y för att undvika stora hopp i styrsignal vid ändring av referensvärdet. Styrsignalen delas upp i tre termer:

$$U(t) = P(t) + I(t) + D(t)$$

där

$$P(t) = KE(t)$$

$$I(t) = \frac{K}{T_i}\int_{t_o}^{t} E(s)ds$$

$$D(t) = -KT_d\frac{dY(t)}{dt}$$

Sätt det senaste samplingstillfället till n och föregående samplingstillfälle till n-1 osv.

Då erhålles direkt:

$$P_n = KE_n$$

För integraldelen gäller:

$$\frac{dI}{dt} = \frac{K}{T_i}E(t)$$

Denna kan approximeras med en differensekvation:

$$\frac{I_n - I_{n-1}}{t_n - t_{n-1}} = \frac{K}{T_i}E_n$$

Då $t_n - t_{n-1}$ = samplingstiden = h erhålles:

$$I_n = I_{n-1} + \frac{Kh}{T_i}E_n$$

Vilket ochså kan skrivas (om $I_o = 0$):

$$I_n = \frac{Kh}{T_i}\sum_{1}^{n} E_m$$

Motsvarande differensapproximation ger:

$$D_n = -KT_d\frac{Y_n - Y_{n-1}}{t_n - t_{n-1}} = -\frac{KT_d}{h}(\ Y_n - Y_{n-1}\ )$$

För att erhålla en lämplig algoritm för en digital regulator, är det önskvärt att skriva styrlagen på formen:

$$U_n - U_{n-1} = t_o R_n + t_1 R_{n-1} - (s_o Y_n + s_1 Y_{n-1} + s_2 Y_{n-2})$$

Skilnaden i styrsignal tecknas då på följande sätt:

$$U_n - U_{n-1} = K(E_n - E_{n-1}) + \frac{Kh}{T_i} E_n - \frac{KT_d}{h}(Y_n - 2Y_{n-1} + Y_{n-2})$$

Om E = R - Y insättes i denna ekvation, kan koefficienterna identifieras.

Integratordelen är i princip en summa av alla tidigare värden, vilket innebär att den fortsätter summera även om utsignalen har bottnat (integratormättning). För att råda bot mot detta måste integraldelen nollställas när signalen bottnar.

Uttryckt i Fortran-kod blir algoritmen följande:

```
C-----------------------------------------------------------------------

C       Bestäm parametrar i polynom av
C       bakåtskiftesoperatorn:
C       (Proportionalitetskonstanten tillkommer i
C       regleralgoritmen)
        if (DPAR.le.0) DPAR=0
        if (IPAR.le.0) then
            T0=1
            S0=1+DPAR/SAMPTID
        else
            T0=1+SAMPTID/IPAR
            S0=1+SAMPTID/IPAR+DPAR/SAMPTID
        endif
        T1=-1
        S1=-1-2*DPAR/SAMPTID
        S2=DPAR/SAMPTID

C       Regleralgoritm:
        VALUT=VALUT+(REFIN*T0-VALIN*S0-OLDVAL1)*PPAR

C       Anti-integratormättning:
        if (VALUT.gt.MAX) VALUT=MAX
        if (VALUT.lt.MIN) VALUT=MIN

C       Styr ut erhållet värde:
        call STYRUT(VALUT)

C       Uppdatera variabler i "minnet":
        OLDVAL1=-T1*REFIN+S1*VALIN+S2*OLDVAL2
        OLDVAL2=VALIN

        return
        end

C-----------------------------------------------------------------------
```

Vid variabla parametrar, kan man före första raden i ovanstående algoritm tillfoga

c. Bestäm parametervärden:

```
Call TIMER (SAMPTID, .....)

PPAR = PROP (VALIN, .....)

IPAR = INTEG (VALIN, .....)

DPAR = DERIV (VALIN,.....)
```

Referenser:

Åström: Reglerteori. [5]

Elmqvist m.fl.: Datorer i reglersystem.
Realtidsprogrammering. [3]

Åström/Wittenmark: Computer Controlled Systems [4]

4 DATALOGISK PROBLEMANALYS

4.1 Lösning av realtidsfunktionerna i VAX- och PDP-FORTRAN

En av grundideerna bakom SIP-systemet är att samma programvara skall kunna användas i VAX- och PDP-datorn. Därför samlas alla maskinspecifika rutiner i en modul som är specifik för respektive dator, men anropsnamn och alla parametrar är identiska. Detta innebär i princip att man simulerar en körning på PDP vid körning på VAX-datorn.

Exempel på några av de sålunda skapade rutinerna är:

```
        subroutine XTIMER (NUM,DELTA,STAT)
```

Denna rutin sätter eventflag nr NUM efter att ha väntat DELTA*20 ms. STAT visar om det gick bra eller inte.

UTFöR I PDP:

```
call MARK (NUM,DELTA,1,STAT)
```

UTFöR I VAX:

```
integer SYS$SETIMR
integer*4 DELTAT(1:2)
DELTAT(1)=-200000*DELTA
DELTAT(2)=-1
STAT=SYS$SETIMR(%val(NUM),%ref
1(DELTAT(1)),%VAL(0),%VAL(0))
```

```
        subroutine XCRGEF (STAT)
```

Denna rutin skapar globala eventflags. 65-80 skapas.

UTFöR I PDP:

```
call CRGF (192,STAT)
```

UTFöR I VAX:

```
integer SYS$ASCEFC
integer*2 PROT,PERM,EFNR
character*4      NAME
include '($SSDEF)'
PROT=0
PERM=0
EFNR=65
NAME = 'EF65'
STAT = SYS$ASCEFC (%VAL(EFNR),
1%DESCR(NAME),%val(0),%val(0))
        :
        :
        :
        :
EFNR=80
NAME = 'EF80'
STAT = SYS$ASCEFC (%VAL(EFNR),
1%DESCR(NAME),%val(0),%val(0))
```

---

subroutine XSETEF (NUM,STAT)

Denna rutin sätter eventflag NUM. Flaggans tidigare värde returneras i STAT.

UTFÖR I PDP:

```
call SETEF (NUM,STAT)
```

UTFÖR I VAX:

```
integer SYS$SETEF
include '($SSDEF)'
STAT = SYS$SETEF (%VAL(NUM))
```

---

subroutine XCLREF (NUM,STAT)

Denna rutin nollställer eventflag NUM. Flaggans tidigare värde returneras i STAT

UTFÖR I PDP:

```
call CLREF (NUM,STAT)
```

UTFÖR I VAX:

```
integer SYS$CLREF
include '($SSDEF)'
STAT = SYS$CLREF (%VAL(NUM))
```

---

integer function XWSCLR*2

Denna funktion ger det värde på parametern STAT som motsvarar att en EF vad nollställd före anrop av XCLREF.

UTFÖR I PDP:

```
XWSCLR=0
```

UTFÖR I VAX:

```
XWSCLR=1
```

---

subroutine XWEFOR (NUM1,NUM2,STAT)

Denna rutin väntar på att en eller flera eventflags skall bli satta.

UTFÖR I PDP:

```
call DECLAR(STAT)
call WFLOR (NUM1,NUM2)
```

UTFÖR I VAX:

```
integer*4 LNUM
integer SYS$WFLOR
include '($SSDEF)'
LNUM=0
LNUM=2**(NUM1-64)
LNUM=2**(NUM2-64)+LNUM
STAT = SYS$WFLOR (%VAL(NUM1),
1%VAL(LNUM))
```

---

subroutine XWEF (NUM,STAT)

Denna rutin väntar på att en eventflag skall bli satt.

UTFÖR I PDP:

```
call DECLAR(STAT)
call WAITFR (NUM,STAT)
```

UTFÖR I VAX:

```
integer SYS$WAITFR
include '($SSDEF)'
STAT = SYS$WAITFR (%VAL(NUM))
```

subroutine XMODE (STAT)

Denna rutin markerar om man kör i VAX eller i PDP.

UTFöR I PDP:	UTFöR I VAX:
STAT=0	STAT=1

Med hjälp av dessa rutiner kan man bygga upp sitt realtidssystem.

Det är då lämpligt att strukturera systemet i moduler, som var och en har en klart definierad uppgift. "Hjärtat" i systemet blir då en processmodul, kallad CLOCK, som administrerar de övriga, genom att efter angiven väntetid, och i tur och ordning aktivera de övriga processerna.

I CLOCK ingår en parameter LDELTA, som markerar systemets "grundfrekvens", d.v.s. antalet cykler i datorns systemklocka som CLOCK själv skall vänta innan den utför en ny loop. Enligt förutsättningarna blir LDELTA = 2.

För synkronisering, skall varje process-modul ha en "egen" unik eventflag definierad. Även CLOCK får en sådan, men den skall ej vara åtkomlig för någon annan process. För övriga processer i systemet reserveras eventflag med nummer 65-75. Då endast 5 av dessa används i systemets nuvarande utformning finnes här en möjlighet till utbyggnad.

För att varje process skall kunna ställa sin samplingstid, skapas en systemglobal vektor HWEWTM (Hard Ware Environment Wait Time).

CLOCK fungerar då så att den sätter eventflag 65-75 i tur och ordning med periodtiden LDELTA*HWEWTM(eventflag). HWEWTM(eventflag) = 0 medför att denna eventflag aldrig sätts.

För att fel ej skall uppstå i samband med användning av gemensamma variabler, t.ex. att en process går in och ändrar värden samtidigt som en annan process använder dessa, brukar man kräva ömsesidig uteslutning, vilket innebär att en process reserverar ett COMMON-block under tiden den använder detta och därigenom utestänger de andra från access till detta block. Det går dock att lösa detta problem genom att i systemet endast tillåta att en specifik process har rätt att skriva i ett visst minnesutrymme. För att detta skall fungera, och att risken för misstag skall minskas, uppställs i SIP-systemet följande regler:

* Endast CLOCK får sätta eventflags.

* Endast den process som "äger" en viss eventflag får nollställa denna samt ändra värdet på "sin" HWEWTM (eventflag).

* Alla variabler i hela systemet följer en namnkonvention där begynnelsebokstaven eller bokstäverna anger vilken typ av variabel det är:

L_____ = Lokal variabel i aktuell programmodul.

G_____ = Gemensam variabel (global) för olika moduler i samma process (internt COMMOM-block, ej installerat som en självständig enhet, utan endast deklarerat i aktuella moduler).

P_____ = Parametrar i ett underprogram.

XXX___ = Gemensam (global) variabel för hela systemet, där XXX anger namnet på det COMMOM-block variabeln tillhör. Dessa variabler är sorterade i grupper, där endast en speciell process tillåts att ändra på värdet av variablerna inom gruppen.

YYYXXX = Namn på COMMON-block där YYY är CMN för systemglobala block, och GLB för processglobala block.

Undantag från denna namnstandard göres i fallet med variabler som definierar systemets gränssnitt mot hårdvaran, då dessa erhåller namn i anslutning till den hårdvara de representerar (se kap.4.3)

Om man vid en revidering av systemet skulle anse det önskvärt att garantera ömsesidig uteslutning vid access till COMMON-blocken kan detta göras med de i systemet befintliga rutinerna enligt följande algoritm:

```
                .
                .
                .
        INTEGER MUTEX,STAT
        PARAMETER (MUTEX=70)
C       NUMMER PÅ DEN EVENTFLAG SOM
C       ANVÄNDS FÖR ÖMSESIDIG UTESLUTNING.
C       SKALL VARA SKILLD FRÅN DE SOM
C       ANVÄNDS FÖR TIDSSYNKRONISERING.
                .
                .
                .
10      CALL XCLREF (MUTEX,STAT)
        IF (STAT.EQ.XWSCLR) GOTO 20
        CALL XWEF (MUTEX,STAT)
        GOTO 10
20      CONTINUE
                .
                .
                .
C       ANVÄND GEMENSAMMA DATA
                .
                .
                .
        CALL XSETEF (MUTEX,STAT)
                .
                .
                .
```

De i systemet ingående processerna bör ha olika inbördes prioritet för att garantera att t.ex. CLOCK inte blir låst av någon tidskrävande och mindre viktig process. Denna tilldelning av prioriteter utförs via referenser i källkodsfilerna av de automatgenererade kommandofilerna. (se kap. 4.2 och kap. 7)

Referenser:

Elmqvist m.fl.: Datorer i reglersystem.
Realtidsprogrammering. [3]

4.2 Säkerhet mot programmeringsfel

I ett större sammanhängande system, med olika program, ofta skrivna av olika personer, kan mycket lätt en ändring i någon del, få svåröverblickbara konsekvenser i en annan del. Dessutom blir felsökningen försvårad då man ej vet vilken av systemets komponenter som ej fungerar.

Vissa steg i att minska dessa felkällor är redan nämnda, såsom moduluppbyggnaden, vilken ökar säkerheten under förutsättning att hela systemet testas varje gång någon modul är ändrad, samt namnstandarden (kap. 4.1).

För att garantera att rätt variabelnamn används vid referens till variabel (vilket är en vanlig felkälla vid programering i fortran), är det lämpligt att alla sådana är deklarerade endast på ett ställe, och att man vid utveckling i VAX:en utnyttjar det "dialektala" fortrankommandot IMPLICIT NONE i alla nyskapade moduler. Detta måste avlägsnas före körning i PDP då det ej är definierat där! För att kunna deklarera globala variabler endast på ett ställe, införs filtypen *.INC, som skall innehålla deklarationer av commonblock. I källkodsfilerna (av typen *.FOR), som skall utnyttja dessa common-block infogas ett include-direktiv (se kap. 7.5)

När nya versioner av systemet skall skapas, antingen genom modifiering till nya tillämpningar, eller andra ändringar av

mindre karaktär, vore det fördelaktigt om alla delar av systemet som kan tänkas bli berörda av ändringen kompileras om. Detta kan ske genom utnyttjande av kommandofiler, en möjlighet som förekommer i både VAX och PDP, fast utseendet på dessa varierar något.

För att kunna välja om man skall nybilda hela eller delar av systemet, krävs ett hierarkiskt uppbyggt nätverk av kommandofiler. Då detta nätverk är uppbyggt av likartade filer vore det fördelaktigt, om dessa kunde genereras automatiskt genom att exekvera någon kommandofil som utför detta. Även andra typer av likartade filer, t.ex. filer för igångsättning av en process, förekommer och borde automatgenereras för att undvika skrivfel vid programmering.

Detta åtgärdas genom att SIP-systemet integreras i ett system för automatgenerering av filer och mjukvarudokumentation, kallat SMS (Software Management System). SMS, som presenteras i kap. 5.4 medför att mjukvarusystem byggs upp hierarkiskt med hjälp av referenser i källkodsfilerna, så att i princip hela systemet utom källkodsfilerna automatgenereras.

För att undvika fel i de filer som trots SMS-systemet, måste skrivas "för hand", bör dessa följa ett standardiserat koncept (exempel på detta är redan omnämnda namnstandarden). Ett sådant standardkoncept redovisas i kap. 7.

4.3 Gränssnitt mot hårdvarumiljön

PDP-datorn och RTP-interfacet kommunicerar med digitala signaler via en buss. De olika korten representeras av adresser med numeriska värden.

För att kunna få ett "programmerarvänligt" snitt mot denna buss, bör man på något sätt omvandla adresseringen så att den sköts via underprogram och/eller variabler med lättfattliga namn och funktioner.

De två tänkbara tillvägagångssätten är i princip

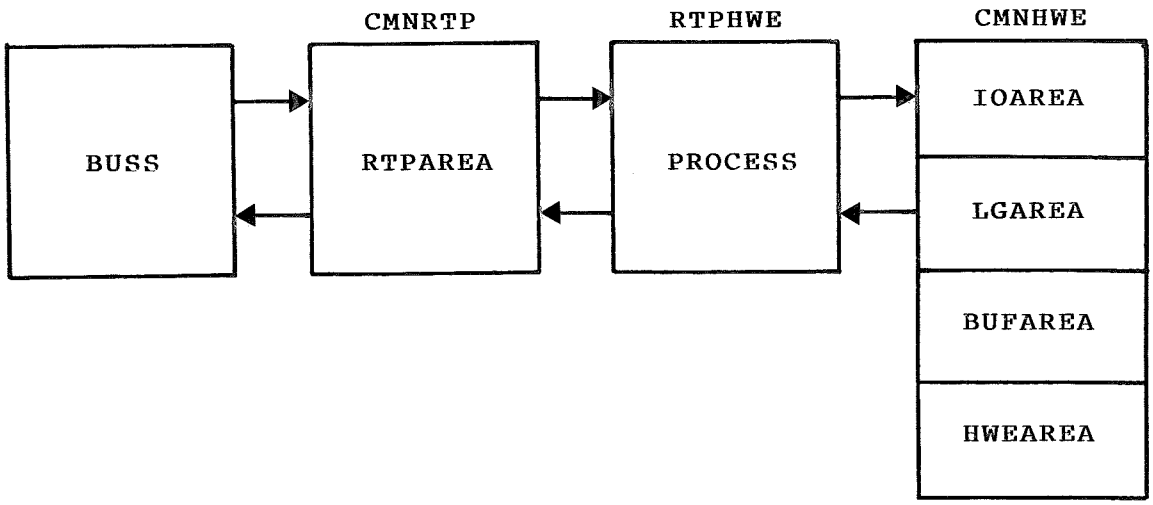
1. Skapa ett bibliotek av underprogram som hämtar in, och styr ut värden, där de olika in- och utgångarna benämns med unika namn. Dessa anropas sedan av systemets processer.

2. Skapa en process som kontinuerligt hämtar in och styr ut värden på alla in- och utgångar med en viss samplingsfrekvens. Denna process skulle då kommunicera med det övriga systemet genom en Common-area.

I kravspecifikationerna ingick att man skulle kunna reglera och lagra mätvärden samtidigt, oberoende av varandra, d.v.s. en process-modul för kontinuerlig insamling av mätvärden behövs ändå. Om man låter samma process även styra ut styrsignalerna, vilket är logiskt, då in- och utsignalerna bör ha liknande gränssnitt gentemot övriga systemet, har man automatiskt erhållit alternativ 2.

Detta ger möjlighet att skapa ett lättfattligt gränssnitt mot övriga systemet genom att döpa variabler i den aktuella common-arean, till namn påminnande om hårdvaru-in-ut-gångarnas beteckningar. Därav den tidigare omnämnda avvikelsen från namnstandarden.

Denna process benämns RTPHWE (RTP-interface-Hard Ware Environment).
Minnesceller för att lagra adress och data som finns på bussen, samt andra nödvändiga variabler för kommunikation mellan RTPHWE och bussen samlas i ett commonblock kallat CMNRTP. Ytterligare ett par commonareor behövs i princip, men har av utrymmesskäl samlats i ett commonblock som representerar RTPHWE:s kommunikation med resten av systemet. Detta döpes till CMNHWE, se Figur 4.1.



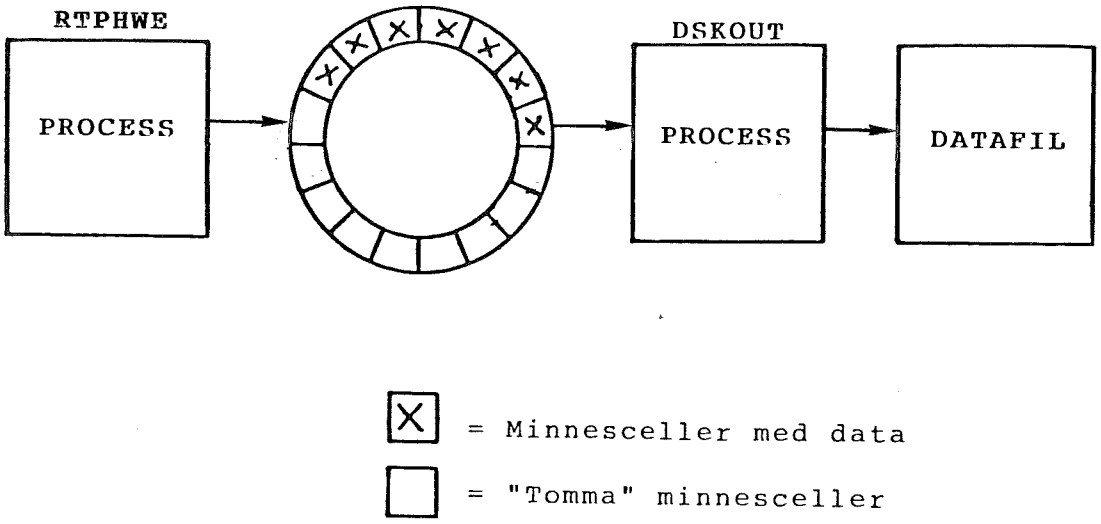
Figur 4.1 Schematisk bild av SIP-systemets gränssnitt mot hårdvaran.

In- och utgångarnas värden lagras i variabler med unika namn, associerade med in- och utgångarnas beteckning, enligt följande:
(se även kap. 2.2):

Beteckning på kortet	Namn på systemglobal variabel	Variabeltyp (Alla är INTEGER)
7436/43	I3643(0:15)	Analog vektor-in
7436/44	I3644(0:7)	Analog vektor-in
7438/20	I3820(0:0) O3820(0:0)	Digitalt ord-in Digitalt ord-ut
7435/82	I3582(0:0)	Digitalt ord-in
7437/37	I3737(0:0)	Digitalt ord-in
7435/81	O3581(0:0)	Digitalt ord-ut
7455/31	O5531(0:3)	Analog vektor-ut
7455/22	O5522(0:0)	Analogt värde-ut

I vissa fall är det fördelaktigt att referera till dessa variabler som en vektor, alltså har dessa minnesceller det alternativa namnet IOAREA(1:34).

Vid lagring av data uppstår problem i och med att denna lagring på skivminne kan ta så lång tid att realtidssystemet inte hinner med under en samplingsperiod. För att kunna lagra värden i den takt som RTPHWE samplar, behövs alltså någon form av buffertlager mellan RTPHWE och den processmodul (DSKOUT) som administrerar lagringen på skivminnet. Buffertar av den här typen, brukar implementeras som en ringbuffert, se Figur 4.2.



Figur 4.2 Ringbuffert.

Ringbufferten deklareras som en vektor (eller flera, då flera värden skall lagras parallellt) kompletterad med variabler som talar om var i vektorn mängden data börjar och slutar (sk. pekare). "Ringen" bildas genom att man vid skrivning och läsning i bufferten fortsätter på första vektorelementet efter att ha passerat det sista. Alla variabler som har med bufferten att göra utgör en del av CMNHWE, där variabelnamnen börjar på BUF- (BUFAREAN i Figur 4.1).

För att spara tid vid lagringen är det önskvärt att ej lagra fler värden än vad som är nödvändigt.
På något sätt bör alltså användaren kunna ange vilka värden som är intressanta att lagra, så att inget onödigt lagras. RTPHWE kan erhålla information om detta via en minnesarea i

CMNHWE. Denna kan då utformas som en vektor där varje element är en flagga som motsvarar ett element i IOAREA. Denna vektor får namnet LGAREA (1:34).

Övriga systemglobala variabler som lagras i CMNHWE (t.ex. samplingstider), följer namnstandarden (HWE-).

4.4 Gränssnitt mot användarna

Programmeraren

Mot programmeraren har inget egentligt gränssnitt definierats, då denna person skall vara kvalificerad att gå in i systemets alla delar och ändra. Det underlättar dock arbetet för denne om de delar som konstruktören ej skall komma åt, är uppbyggda enligt exakt samma principer (standardkoncept, namnstandard) som de delar som konstruktören skriver.

Konstruktören

Då olika processer i ett realtidssystem kommunicerar via common-areor, faller det sig naturligt att det av det "fasta" systemet som konstruktören "ser" är en eller flera common areor.

En av dessa blir den tidigare nämnda CMNHWE, där hårdvaran alltså ur konstruktörens synvinkel motsvaras av integervariabler som kan anta värdena -2048 - +2047 (se kap. 2.2). Via CMNHWE kan konstruktören också ge instruktioner om samplingstider och lagring av mätdata.

Vid utformningen av en tillämpning skulle det gå att erhålla ett fungerande system enligt angivna principer bara genom att göra ett applikationsprogram som arbetar mot CMNHWE (i realtid), men vid utformning av sådana applikationsprogram finner man snart att den allra största delen av programmeringsarbetet och programkoden går åt till operatörskommunikation. P.g.a. att

utskrift på skärmen är en mycket långsam procedur jämfört med andra aktiviteter i systemet är det lämpligt att lägga operatörskommunikationen i en egen process, så att dess samplingstid blir oberoende av den som krävs för regleringen (eller annan applikation). Ovanstående leder lätt till slutsatsen att man om man standardiserar displayens utseende, kan frikoppla operatörskommunikationen från konstruktörens ansvarsområde, och infoga denna process som en del av det "fasta" systemet. Det är dock nödvändigt att ge konstruktören en möjlighet att inom givna ramar, utforma en individuell display för varje tillämpning. Detta löses praktiskt genom att OPCOM-processen, när den startas läser en datafil, skriven av konstruktören, där instruktioner om hur displayen skall utformas finns.

Gränssnittet mellan OPCOM-processen, och applikationsprogrammet utgörs då på vanligt sätt av ett common-block (CMNOPC). I detta block reserveras olika celler för kommunikation i olika riktningar, i enlighet med kap. 4.1. Dessa celler blir även föremål för standardisering då deras antal är begränsat (se kap. 7.4). Variablernas unika nanmn får lösas genom kommentarer på dataskrämen och i programkoden.

Vid programutveckling är det lämpligt om man utnyttjar datorernas operativsystem för att dra en gräns för vad som är tillåtet och otillåtet att ändra på. Dessa operativsystem är uppbyggda så att programvara är samlad i filer av olika typer (tex. *.FOR, *.COM). Filerna är samlade i kataloger (directory), vilka i sig utgör filer i den aktuella katalogens huvudkatalog, så att all programvara är samlad i en trädstruktur (se Fig. 4.3).

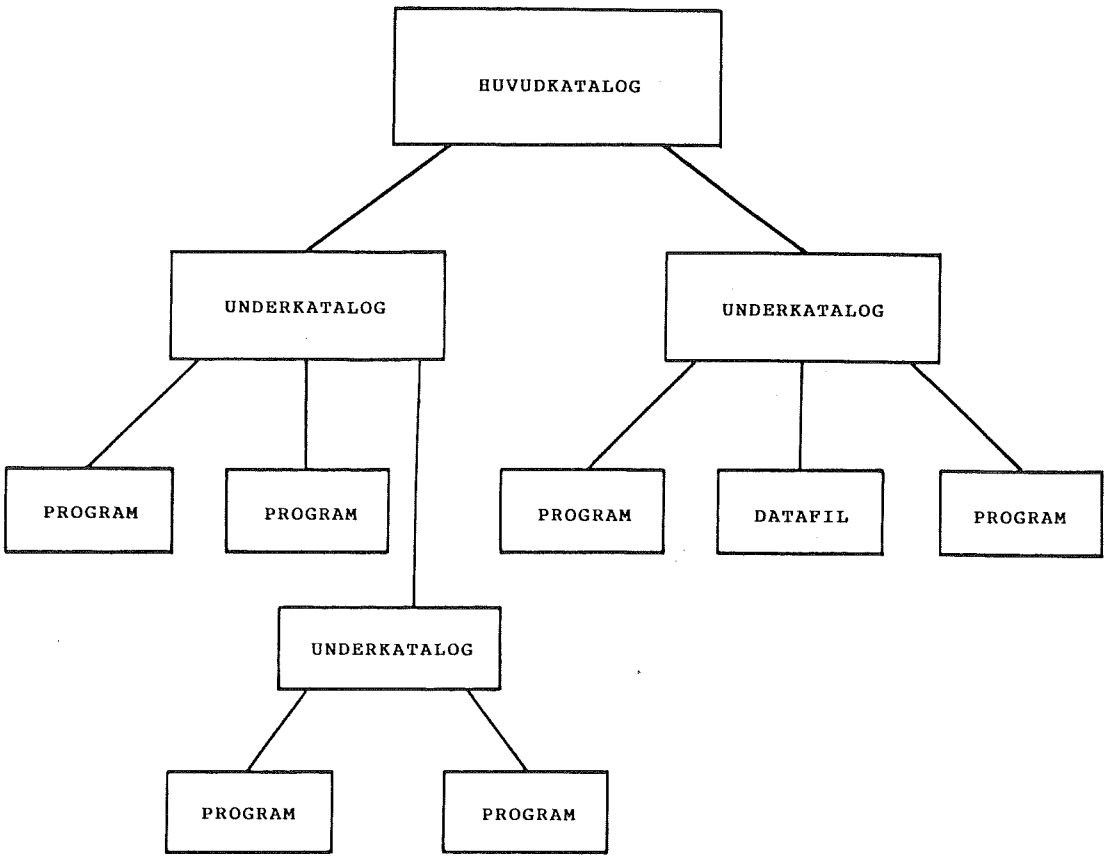


Fig. 4.3 Operativsystemens trädstruktur.

Denna kan då utnyttjas så att "samhörande" filer samlas i underkataloger för sig. Detta är i SIP-systemet utfört så att de filer som konstruktören skall komma åt och de som endast programmeraren skall hantera, är samlade i olika parallella kataloger. Dessutom skapas en ny parallell katalog för varje tillämpning som skapas.

Operatören

Gränssnittet mot operatören utgörs av terminalens tangentbord och bildskärm (samt kringutrustning). För att göra systemet säkert mot felaktiga nedtryckningar, bygger man OPCOM-processen så att den endast reagerar för de tryckningar som ger meningsfull information till OPCOM. Det skall ej heller gå att skriva

något på skärmen, utom på de platser som konstruktören tillåter, via sina display-instruktioner.

För att få kontroll över detta, måste all kommunikation mellan terminal och dator gå via OPCOM-processen. Övriga systemet har endast kontakt via CMNOPC-blocket. För att ge möjlighet åt applikationen att erhålla kommando via tangentnedtryckningar, måste OPCOM lagra information om dessa i CMNOPC. Denna information kan överföras på ett flertal sätt, här har valts att låta minnescellen OPCKEY innehålla ASCII-koden av senast nertryckta tangent. Undantag för detta görs av de tangenter som innebär direkta kommandon till OPCOM. För att endast kunna skriva på skärmen på vissa fördefinierade platser, flyttas cursorn på skärmen endast mellan dessa platser med ett kommando (tangent "blank"). För att ytterligare gardera sig mot felskrivningar, skickas inga värden till CMNOPC (undantag OPCKEY) förrän på givet kommando (tangent "s").

För kommunikation åt det andra hållet läser OPCOM med givet samplingsintervall utdata från CMNOPC. Detta skall då presenteras för operatören på skärmen. Då kontinuerlig utskrift vid varje samplingstillfälle stjäl mycket arbetstid hos datorn, bör man ej skriva ut fler värden än nödvändigt. Detta löses genom att endast av konstruktören angivna minnesceller i CMNOPC, presenteras på skärmen.

För grafisk persentation är det ännu viktigare att göra så lite som möjligt, varför ett maximalt antal grafiskt presenterade värden bör anges. Vid den tillämpning (LINDA)

till vilken SIP-systemet från början är konstruerat, behöver man 4 grafiska kurvor (börvärde + ärvärde för varje rullställ), och en rimlig bedömning är att de flesta tänkbara tillämpningar klarar sig med detta. Alltså väljes maximalt antal grafiska kurvor till 4. För att minimera programmeringsarbetet för konstruktören utnyttjas för detta ett standardutformat grafiskt "fönster" (endast ett), där endast storleken på det rektangulära fönstret kan påverkas (av konstruktören).

4.5 Lagring av mätdata

I kapitel 4.3 beskrivs konstruktionen med ringbuffert för överföring av mätdata från RTPHWE till den del av systemet som administrerar lagringen av mätdata. Den komplettering som behövs för lagringen är en processmodul (DSKOUT) som läser från bufferten och skriver ut detta på en datafil.

Även med en ringbuffert i systemet, finns behov av att optimera snabbheten hos DSKOUT; för att minska risken att bufferten blis full. Det som tar lång tid, är att skriva på skivminnet, och att omvandla data från hårdvaran i binärkod till ASCII-kod. Snabbaste skrivhastigheten uppnår man genom att låta DSKOUT skriva data direkt i binärkod.

Detta skapar ett behov av att i SIP-systemet integrera ett program, BINASC, som omvandlar en fil i binärkod till en fil med motsvarande data i ASCII-kod, samt ett program, ASCBIN, som omvandlar från ASCII-kod till binärkod.

För att operatören skall kunna styra lagringen krävs kommunikation mellan OPCOM och DSKOUT. Detta sker genom att OPCOM skriver en numerisk kod i cellen HWELST i CMNHWE. CMNOPC är endast avsedd som gränssnitt mellan OPCOM och applikationen, så för att meddela operatören vilken status som gäller för lagringen, måste detta ske genom att applikationen läser i en för detta ändamål avsedd cell i CMNHWE, kallad HWELSU, vars värde kan anta en delmängd av de värden som kan förekomma i HWELST:

0 = Vila (datafil ej öppen)

2 = Lagring pågår

3 = Paus (datafil öppen)

Den lagrade mätdatan kan utnyttjas för att reproducera ett händelseförlopp, som utspelats i laboratoriemiljön. För att utföra detta krävs en modul som gör samma sak som DSKOUT, fast "bakvänt", dvs läser från datafil till ringbufferten, vilken följdriktigt döpes till DSKINP. Observera att denna process till skillnad från de övriga ej utgör en oändlig loop, utan processen terminerar när indata från filen har tagit slut.

För att sedan åstadkomma en simulering av händelseförloppet krävs att processen RTPHWE ersätts av en annan process (RTPSWE), som istället för att hämta sina mätvärden från hårdvaran (RTP-interfacet) hämtar motsvarande information från ringbufferten, och placerar i aktuella minnesceller i IO-arean.

4.6 Strukturering av systemet

I de föregående avsnitten har olika moduler i systemet och deras inplacering i operativsystemet presenterats. Hur SIP-systemet byggs upp av dessa moduler ska diskuteras i detta avsnitt.

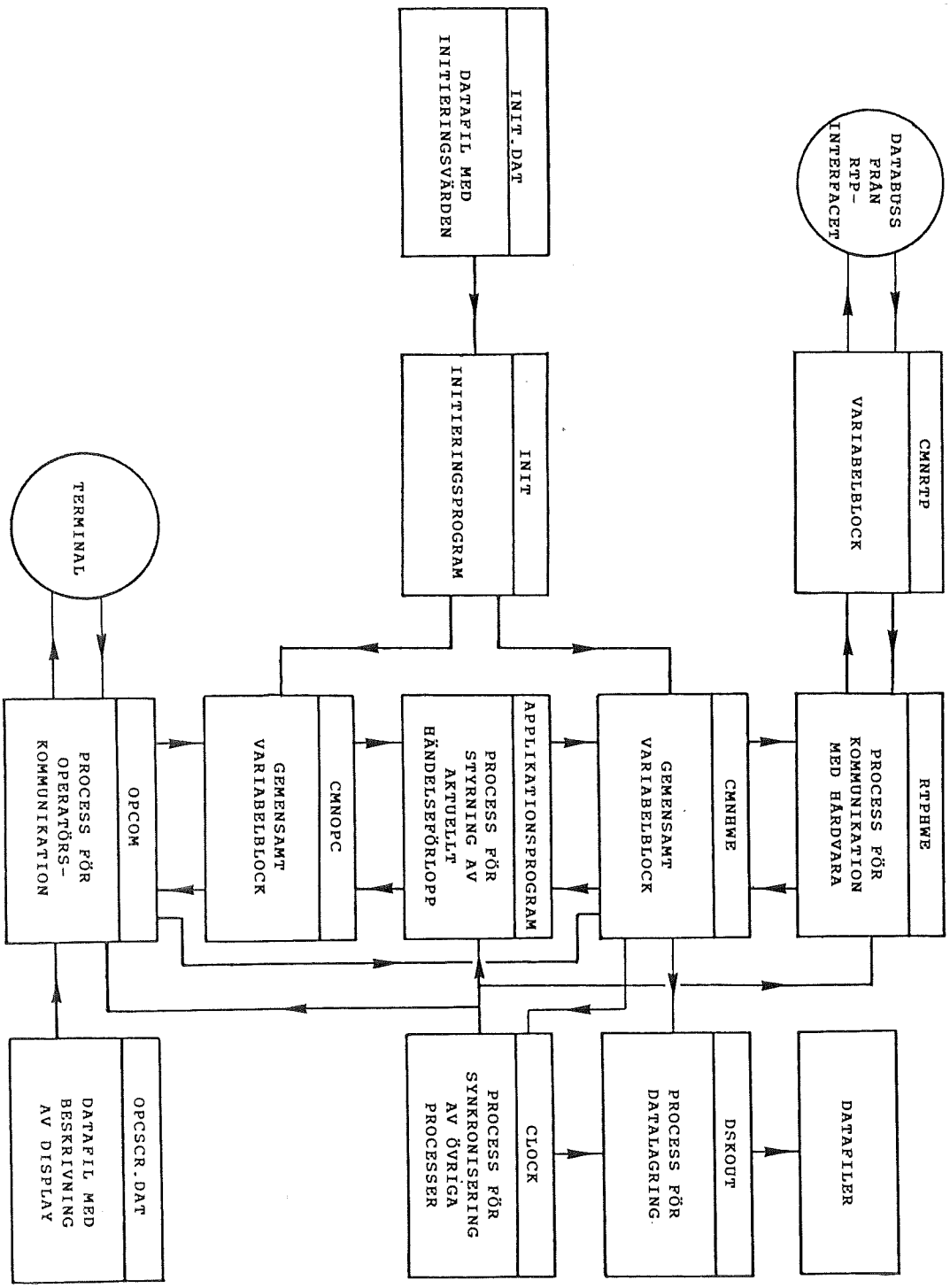
Utöver de moduler som redan nämnts, måste systemet kompletteras med moduler som hanterar igångsättningen av hela systemet i för varje tillämpning aktuell version.

Datorernas operativsystem ger här en bra möjlighet i form av kommandofiler. I varje katalog som utgör huvudkatalog för någon användaridentitet på datorn, finns en standardkommandofil LOGIN, som automatiskt exekveras varje gång man påbörjar

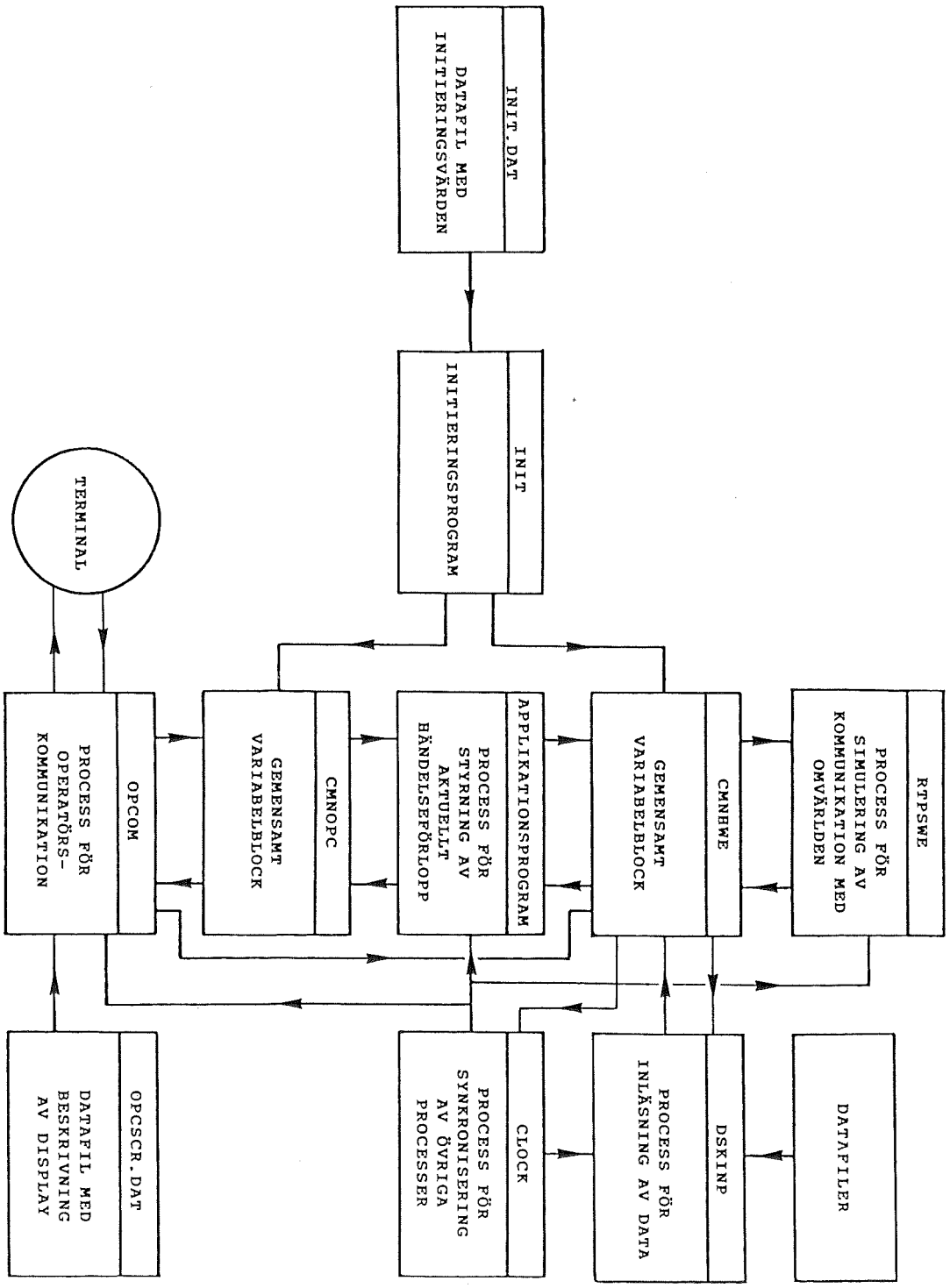
en körning på datorn. Detta ger möjlighet till en för operatören mycket enkel igångsättningsprocedur. Om man låter katalogen innehållande moduler specifika för en viss applikation, vara huvudkatalog för en användaridentitet med applikationens namn och samma namn som password, kan man låta LOGIN innehålla instruktioner för igångsättning av de programkörningar och processtarter som krävs vid denna applikation. För att undvika att operatören av misstag ändrar något bland modulerna i denna katalog, bör LOGIN avslutas med kommandot LOGOUT, vilket innebär att man automatiskt avslutar hela programkörningen när processerna har terminerat.

Ett behov av att kunna placera default-värden i common-areornas minnesceller förekommer också. T.ex. måste man någonstans ange vilka filnamn som gäller vid läsning/skrivning i datafiler av processerna. Detta bör alltså utföras som en första åtgärd i LOGIN. Den enklaste lösningen är att helt enkelt köra en programmodul med denna uppgift, kallad INIT, som läser instruktioner från en standardfil i varje applikationskatalog, kallad INIT.DAT.

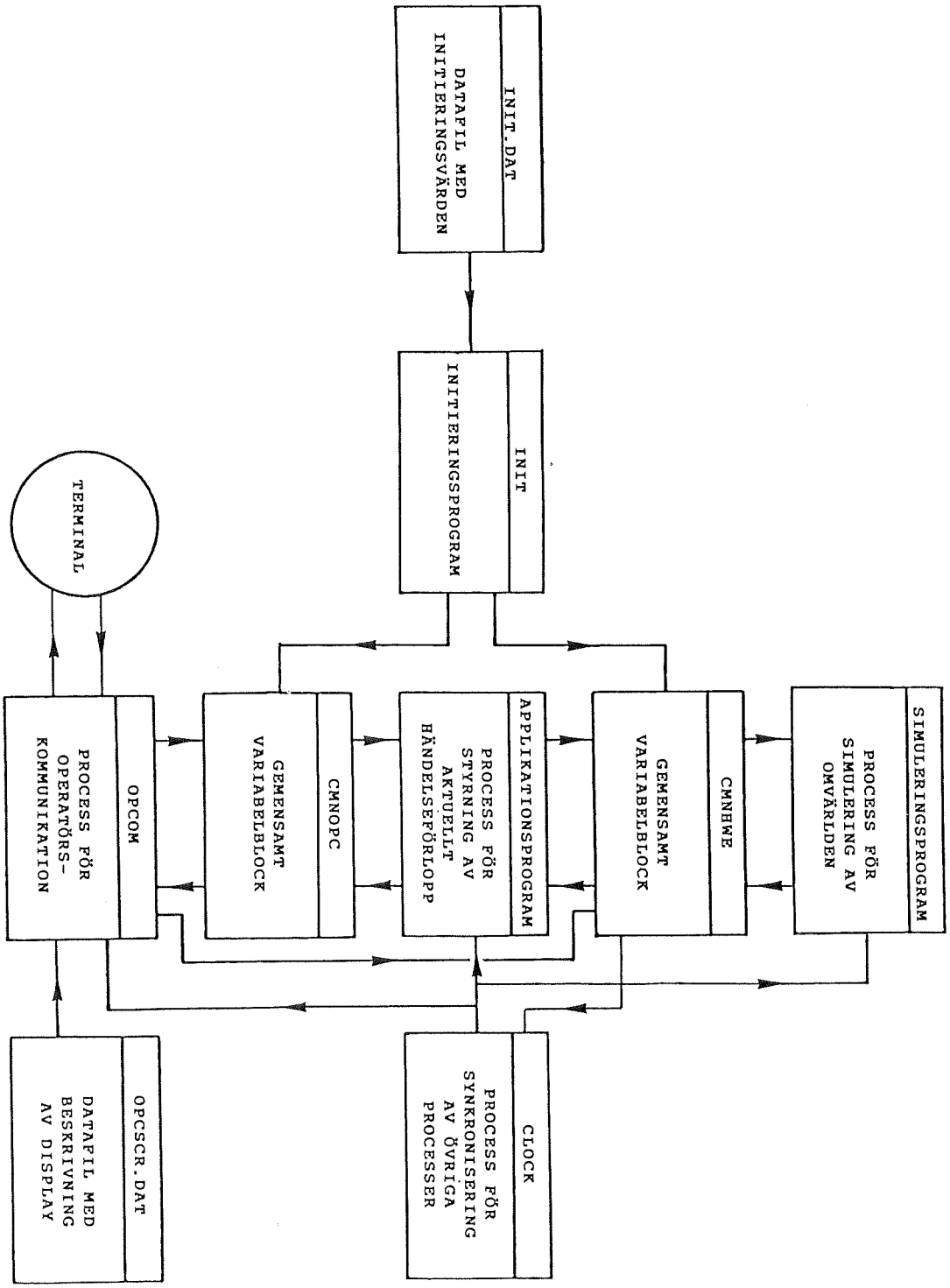
Med LOGIN och de tidigare presenterade modulerna finns allt som behövs för grundversionen av SIP-systemet. En sammanställning av systemets struktur göres enklast i grafisk form, se Figur 4.4, 4.5 och 4.6.



Figur 4.4 SIP- systemet vid körning i PDP-datorn.



Figur 4.5 SIP-systemet vid rekonstruktion av verkligt händelseförlopp i VAX-datorn.



Figur 4.6 SIP- systemet vid körning av simuleringsprogram i VAX-datorn.

5 PRESENTATION AV SIP-SYSTEMET

5.1 Inledning

Det realtidssystem som har framkommit ur resonemangen i tidigare kapitel har alltså erhållit namnet SIP. Detta namn är en förkortning av en beskrivning av vad systemet utför:

> SIP = ett systemt för att utföra Styrning av någon försöksuppställning. Insamling av mätdata från denna försöksuppställning. Presentation av mätdata och styrdata på grafisk terminal, under pågående styrning av försöksuppställningen.

SIP-systemets mjukvarustruktur framgår av Figurerna 4.4 - 4.6, där den framställs i form av blockschema. Det är utvecklat för att (med smärre modifieringar) passa in i Digital Equipments olika datorer av typ VAX och PDP, och i sin helhet skrivet i Fortran + datorernas kommandospråk. Det är uppbyggt för att vara väl anpassat för kommande förändringar, och borde ha förutsättningar att överleva vid eventuellt framtida byte av datorutrustning på TFL.

5.2 Inplacering i datorernas operativsystem

Systemet är placerat i PDP-datorn på två diskenheter, som var och en innehåller en komplett uppsättning av systemet, så att en fungerande version alltid finns att tillgå under utveckling av programvaran. De bägge kopiorna av systemet finns i katalogerna: DM0:<300,310> och DM1:<300,310>.

Applikationsmodulerna finns i kataloger med nummer:

<300,311>, <300,312> o.s.v.

För att göra flyttningen av systemets programvara i sin helhet möjlig, har systemet placerats i VAX:en i två versioner, med logiska namn på diskenheterna och katalogerna, lika lydande med dem som gäller i PDP.

I VAX-datorn kan man utnyttja möjligheten att ge logiska namn, så att man flyttar sig mellan de parallella systemen med kommandona:

$ DRF Flyttning till det system som är avsett att alltid vara senaste komplett fungerande version av det totala systemet, d.v.s. det är här som simulering utförs.

$ UTV Flyttning till det system som är avsett för programutveckling.

Inom varje system kan man på VAX:en flytta sig mellan katalogerna med följande kommandon:

$ SIP Flyttning till den katalog som innehåller de fasta modulerna i SIP-systemet.

$ Applikationsnamn Flyttning till den katalog som innehåller de moduler som är specifika för en applikation.

Som alternativ till det numeriska katalognamnet, kan man på VAX:en använda SIP, $Applikationsnamn$...

När man vill överföra ett fungerande system från UTV- till DRF-systemet, exekverar man kommandofilen:

```
$ É$SIP$:UTVTODRF
```

5.3 Inbyggd dokumentation

En fullständig dokumentation av SIP-systemet med alla ingående delar, finns lagrad i datorn i filen SIP:SIP.LIZ, för att kunna hålla denna uppdaterad vid eventuella förändringar. En ny version av denna dokumentation genereras automatiskt med hjälp av SMS-systemet (se kap. 5.4) via kommandot:

```
$  SMS/DOCUMENT  $SIP$:SIP.TXT/OUTPUT=$SIP$:SIP.LIZ
```

En ny version av denna dokumentation skapas också varje gång man exekverar kommandofilen TOTALBYGG.COM, vilken med hjälp av SMS (se kapitel 5.4) bygger upp hela systemet, se vidare i kapitel 7.7.

5.4 SMS-systemet

SMS står för Software Management System och är ett program-utvecklingsverktyg där ett system av moduler kan byggas upp i en hierarkisk struktur, inkluderande i datorn placerad dokumentation av hela systemet.

SMS är utvecklat och marknadsförs av OPIAB-KONSULT. Vid utvecklingen av SIP-systemet har SMS-systemet installerats i

TFL:s VAX-dator för att användas som ram till SIP-systemet. I dokumentationen för SIP i datorn (filen SIP:SIP.LIZ) finns en utförligare beskrivning av SMS-systemet.

SMS innebär följande:

STRUKTUR I alla källkodsfiler som skrives införs en referens, som placerar denna fil i en hierarkisk trädstruktur omfattande hela (SIP-)systemet. Vilka referenser som måste göras vid utveckling av nya tillämpningar framgår av kap. 7 samt dokumentationen av SIP-systemet i datorn.

DOKUMENTATION: I källkodsfilerna kan man skriva in kommentarrader, vilka markeras med %DOC%, %DOCBEG% och %DOCEND%, se kap. 7.5. Dessa medtages sedan vid en automatgenerering av dokumentation av hela den hierarkiska programstrukturen.

AUTOMATGENERERING: SMS innehåller funktioner för att automatgenerera (källkods- och kommando-) filer som i systemet har ett standardiserat utseende, så när som på givna parametrar (t.ex. Applikationsnamn).

5.5 Definition av filtyper

I VAX- och PDP-datorns operativsystem skall varje filnamn kompletteras med en filtyp på tre bokstäver. Förutom de typer som är standard i operativsystemen, definieras i SIP-systemet ytterligare några filtyper som är standard i SIP-systemet. Här följer en sammanställning av i SIP-systemet förekommande standardfiltyper.

FOR: Källkodsfil i programspråket Fortran.

INC: Källkodsfil i Fortran, som ej utgör en självständig modul, utan skall infogas i (en eller) flera olika FOR-filer med ett include-direktiv.

DAT: DATAFILER avsedda att läsas av någon programmodul.

TXT: Textfiler ingående i dokumentation av systemet (utöver det som finns i källkodsfilerna i form av kommentarer).

BIN: Binär datafil skriven av programmodul.

ASC: Datafil i ASCII-kod, skriven av programmdoul.

COM: Kommandofil på VAX-dator.

CMD: Kommandofil på PDP-dator.

CMV: Automatgenererad kommandofil för kompilering och länkning i VAX.

CMP: Automatgenererad kommandofil för kompilering och länkning i PDP.

DUM: Dummy-fil = tom fil, avsedd för tillfällen när referens till filnamn behövs utan att filen uppfyller någon funktion.

STV: Automatgenererad kommandofil för start av process i VAX.

STP: Automatgenererad kommandofil för start av process i PDP.

LNV: Automatgenererad kommandofil med länkningskommandon i VAX.

LNP: Automatgenererad kommandofil med länkningskommandon i PDP.

NOD: Fil som beskriver en nod i SMS-systemets trädstruktur.

MAP: Automatgenererad fil vid kompilering och länkning.

OBJ: Kompilerad källkodsfil.

EXE: Exekverbart program i VAX, bildat av OBJ-filer genom länkning.

TSK: Exekverbart program, bildat av OBJ-filer i PDP.

LIS: Programlista, bildad vid kompilering.

BYG: Automatgenererad fil för byggning av systemets automatgenererade filer.

BTC, DOV, DOX, DOY, DOZ, MEC, RNT:
Mellanled vid automatgenerering.

LIZ: Färdig automatgenererad dokumentation.

5.6 Presentation av ingående filer

Här följer en kort presentation av de i SIP-systemet ingående fasta moduler, utom de som hanterar SMS-systemets automatgenerering av filer och dokumentation eller själva är automatgenererade. En liknande presentation av applikationsspecifika moduler presenteras i kap. 7.1. Med sekventiellt program menas program som ej löper i oändlig loop (=process).

ASCBIN Sekventiellt program, omvandlar en fil med data lagrad i ASCII-kod till en fil med motsvarande data lagrad binärt.

BINASC Sekventiellt program, omvandlar en fil med binärt lagrade data till en fil med motsvarande data lagrad i ASCII-kod.

BUFDRV Bibliotek med underprogram för hantering av ringbufferten.

CLEANSIP Kommandofil som tar bort samtliga automatgenererade filer i systemet.

CLOCK Process som sköter synkroniseringen med klockan i systemet. Processen går i en loop och sätter eventflags

CMNHWE Common block, gemensamt för de delar som arbetar mot och med hårdvara. Innehåller styrsignaler till processer som CLOCK, RTPHWE och DSKOUT, (eller motsvarande) in-utgångar på RTP-interface, samt en ringbuffert.

CMNOPC Common block, innehållande alla gemensamma datastrukturer för applikationsprogram och OPCOM.

CMNRTP Common block, som beskriver gränssnittet mot RTP-interface. Variablerna här är ej kopplade till vanliga minnesceller, utan till adressregister för hantering av RTP-interfacets buss. Endast den process som skall hantera kommunikation med interface är kopplad till dessa variabler.

DELMNT Kommandofil som används i PDP för att installera en ny version av systemet från flexskiva. Tar bort det gamla systemet och exekverar sedan kommandofilen RX2PDP.

DSKINP Process som läser data från en fil, och placerar denna data i ringbufferten i CMNHWE.

DSKOUT Process som läser data från ringbuffert i CMNHWE och placerar denna data i en fil.

DUMMY Tom fil som används för att kunna utnyttja SMS-referenser för att ge inbördes prioritet åt de olika processerna.

E Kommandofil för PDP-datorn som innebär att man hamnar i EDT-editorn med samma kommandon definierade som i VAX:en där kommandot E innebär att EDT-editorn aktiveras med vissa ingångsparametrar. OBS! att i bägge datorerna måste filnamn anges efter E:et.

EDTINI Datafil med initieringsdata för EDT-editorn.

GLBINI Lokalt common block för programmet INIT

GLBOPC Lokalt common block för processen OPCOM.

INIT Sekventiellt program som läser data från filen INIT.DAT i applikationens kataloq (se kap. 7.3) och i enlighet med instruktioner där, initierar common-areorna i systemet.

INSCMN Kommandofil som installerar alla globala kommonareor i PDP.

INSPRG Kommandofil som tar bort gamla tasks och installerar nya. Gäller för systemets alla tasks.

LBRPDP Bibliotek av underprogram med för PDP-datorn specifika kommandon, vilka skall kunna anropas med anrop gemensamma för bägge datorerna (se kap. 4.1).

LBRVAX Bibliotek innehållande samma underprogram som LBRPDP, men skrivna i den språkvariant som gäller för VAX.

LBRXXX Dummyfil för att styra automatisk kompilering och länkning mot LBRPDP eller LBRVAX, beroende på vilken dator man befinner sig i när detta utförs.

OPCOM Process som kan presentera variablers värden kontinuerligt på bildskärm, både grafiskt och numeriskt, lämnar indata och hämtar utdata från CMNOPC. Läser instruktioner om skärmens utseende från fil angiven i INIT.DAT.

OPCSCR Datafil med instruktioner för bildskärmens utseende, i det fall man vill kunna se värdena på samtliga möjliga minnesceller i CMNOPC.

PDPRX2 Kommandofil för flyttning av hela systemet från PDP-dator till flexskiva.

RTPHWE Process som hämtar värden från RTP-interfacet och placerar dem i IO-arean i CMNHWE, samt placerar de värden som är markerade för lagring i LG-arean i ringbufferten (i CMNHWE). Dessutom hämtas utdata från IO-area och skickas till RTP-interfacet.

RTPSWE Process som hämtar värden från ringbufferten och placerar dessa som indata i IO-area.

RX2PDP Kommandofil för flyttning av systemet från flexskiva till PDP.

RX2VAX Kommandofil för flyttning av systemet från flexskiva till VAX.

STARTUP Kommandofil för initiering vid start av dator.

TOTALBYGG Kommandofil som bygger upp en ny version av alla automatgenererade filer, inklusive dokumentation av systemet, samt exekverar TOTBYG.

TOTBYG Komamndofil som kompilerar och länkar hela SIP-systemet.

UTVTODRF Kommandofil som förflyttar senaste versionen av systemet på UTV (utveckling) till DRF (driftssystemet) på VAX-datorn.

VAXRX2 Kommandofil som flyttar systemet från VAX-dator till flexskiva.

6 INSTRUKTION FÖR KÖRNING AV SIP-SYSTEMET

6.1 Körning av applikation på PDP-11

Antag att tillämpningen (applikationen) HILBERT skall köras. Körning av denna i PDP-datorn görs enligt följande:

1 Kontrollera i dokumentationen av HILBERT vilka in- och utsignaler som skall kopplas till vilka in- och utgångar.

2 Koppla dessa och tillse att signalerna passerar lämpliga filter och förstärkare för att erhålla rätt signalnivå:

För digitala signaler: +5 V är logisk nolla
0 V är logisk etta

Analoga ingångar +-10,21 V

Analoga utgångar +- 5,10 V

3 Slå på strömförsörjningen till datorn, terminalen och RTP-boxen.

4 Tryck på knappen BOOT på datorn.

5 Skriv DM1 (eller DM0) med stora bokstäver, detta för att datorn skall veta var den skall hämta instruktioner för start av systemet.

6 Datorns operativsystem frågar efter tid, svara på detta.

7 Nu är datorn igång och de olika applikationerna kan köras genom att logga in på datorn med applikationernas namn:

Fråga:	Svar:
>	HELLO
Account or name:	HILBERT
Password:	HILBERT

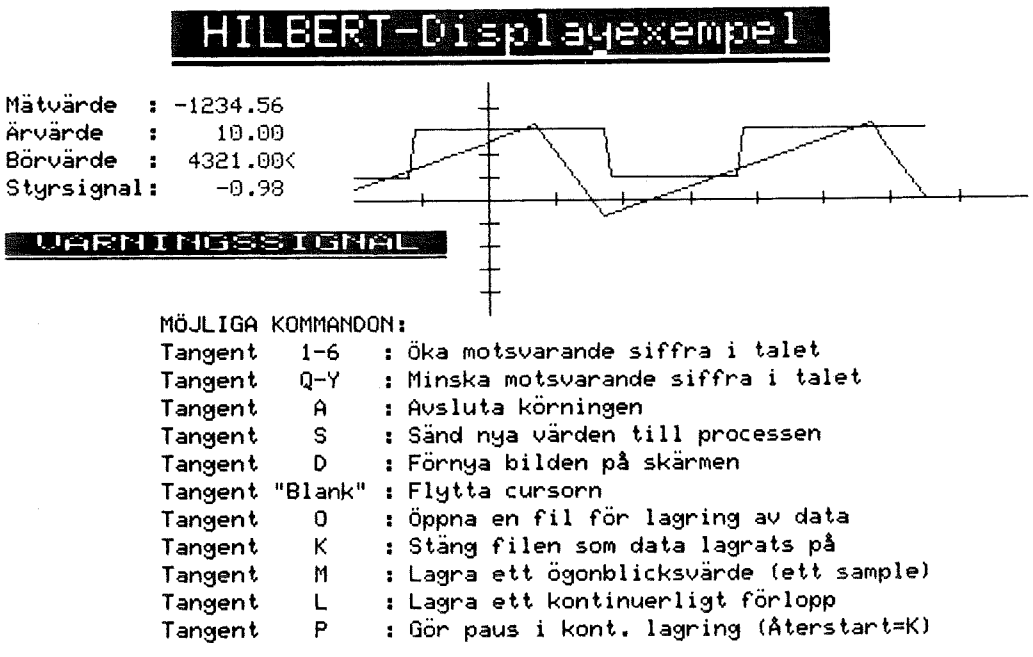
8 Hela systemet startas nu i rätt ordning och om olika alternativa displayer förekommer kommer nu en meny på skärmen med kommandon som kan ges.

```
************ HILBERT, MENY-EXEMPEL ************

        FÖLJANDE ALTERNATIVA REGLERSTRATEGIER
        KAN ANVÄNDAS FÖR ATT STYRA
        DEN SIMULERADE PROCESSEN HILBERT:

        1 PID-regulator
        2 Parameterstyrd PID-regulator
        3 Kalmanfilter
        4 Adaptiv reglering

        FÖLJANDE ALTERNATIVA DISPLAYER
        KAN ERHÅLLAS:

        1 Numeriskt presenterade värden
        2 Grafiskt presenterade värden

VÄLJ REGLERSTRATEGI: 2
VÄLJ DISPLAYVARIANT:
```

Figur 6.1 Exempel på en meny.

Vid enklare tillämpningar kommer displayen upp direkt och möjliga kommandon finns då på displayen. I SIP-systemet ingående standardkommandon framgår av Figur 6.2. Varningssignalen är en händelseindikation, vilket i SIP-systemet innebär en textremsa som skrives ut på skärmen först efter att en viss händelse har inträffat.

De värden på skärmen som representerar data som operatören skall kunna ändra på (t.ex. börvärden, insignaler), ändras genom att flytta cursorn till aktuellt tal, och genom tangenttryckningar på tangenterna 1-6 eller Q-Y för att räkna upp respektive ner någon av talets sex siffror (all indata ges som reella tal mellan +9999.99 och -9999.99 med två decimalers noggrannhet). Värdena ändras endast på skärmen vid denna procedur, den process som utför applikationen arbetar vidare med de gamla värdena tills operatören signalerar att de skall ändras genom att trycka på S-tangenten. Då börjar applikationen arbeta med de värden som fanns på skärmen vid tangenttryckningen.



Figur 6.2 Exempel på en display.

9 Vid avslutad körning, trycker man på tangenten A och återvänder då till eventuell meny. Om meny saknas avslutas körningen direkt och användaren loggas ut på datorn.

10 Datorn är nu beredd att köra en ny applikation eller att stängas av med strömbrytaren.

6.2 Lagring av mätdata

Under körning enl. kap. 6.1, kan man när man vill, lagra undan valfria mätdata på en datafil. För att vidare kunna behandla dessa krävs ett konto på resp. dator.
För allmän användning i samband med SIP-systemet finns kontot:

VAX	PDP
USERNAME: SIP	Account or name: SIP
PASSWORD: SIP	Password: SIP

Instruktioner för vilka mätvärden som skall lagras, och på vilka filer, samt hur snabbt denna lagring skall ske, finnes i filen INIT.DAT.
Detta är bestämt i och med konstruktion av aktuell tillämpning och kan ej ändras under körning.

Om så önskas är det lätt att ändra dessa instruktioner mellan två körningar genom att editera i filen INIT.DAT, i enlighet med kap. 7. En varning bör ges för att ändra fel rader i INIT, den styr mycket annant (i princip hela systemet)!
Någon förnyad kompilering el.dyl. behövs ej efter denna ändring.

Mätdata lagras under pågående körning med följande kommandon:

O = OPEN, innebär att filen för datalagring öppnas, d.v.s. en ny version av denna fil (nytt versionsnummer) skapas, och görs färdig för att mottaga data.

K = CLOSE, innebär att filen stängs, d.v.s. det går ej längre att skriva i denna fil, vid nästa open bildas en fil med nytt versionsnummer. Detta innebär att man vid samma körning kan skapa flera datafiler genom upprepade O, K-kommandon.

L = LOGG, innebär att lagringen startas när L-tangenten trycks ned.

P = PAUSE, innebär att lagringen tillfälligt stoppas. Filen är fortfarande öppen, och lagringen fortsätter efter nästa L-tryckning.

För att spara tid under den pågående körningen, lagras tecknen binärt i filen avsedd för detta. För att överföra denna fil i en fil med läsbara tecken (ASCII-kod), måste man flytta sig till TILLÄMPNINGENS directory (katalog) där man exekverar:

run SIP:INIT (<300,310>INIT på PDP)

detta för att få rätt namn på filerna.

Sedan kör man:

run SIP:BINASC (<300,310>BINASC)

Om man i stället kör:

run SIP:ASCBIN (<300,310>ASCBIN)

omvandlas i stället data i ASCII-kod till data i binär kod.

För att flytta en datafil från PDP til VAX för vidarebehandling där, göres följande:

I PDP

1. Stoppa in en floppy-disk med definierade directoryn (dessa är definierade om skivan använts för flyttning av hela systemet enl. kap. 5) i drivern.
2. Skriv MOUNT DYØ DYØ
3. Skriv COPY filnamn DYØ:filnamn
   (exempel på filnamn: <300,325>HILBERT.BIN)
4. Skriv DISMOUNT DYØ
5. Flytta skivan till VAX.

I VAX

1. Sätt skivan i övre drivern på Vax:en
2. Skriv MOUNT DYAØ: DYØ
3. Skriv COPY DYØ:filnamn filnamn
4. Skriv DISMOUNT DYAØ

6.3 Simulering på VAX 11

Det som här kallas simulering innebär egentligen två väsenskilda förlopp.

1. Körning av samma system, som på PDP-datorn (med den verkliga processen inkopplad) fast med den verkliga processen utbytt mot en indatafil bestående av mätvärden från en verklig körning. Detta innebär att exakt samma förlopp sett ur systemets och operatörens synvinkel åter utspelar sig. Möjlighet finns här att genom att ändra samplingstider för de ingående processerna (se kap. 7) köra förloppet i "slow motion" och dylikt.

2. Körning av simuleringsprogram där algoritmer simulerar den verkliga processens uppförande.

Vilka av dessa varianter som gäller styrs av konstruktören vid skapandet av applikationens startfil (ex. $HILBERT$START.COM) se kap. 7.2.
För att köra en simulering på VAX:en följer man endast samma instruktioner, vilka gäller för en körning mot verklig process i PDP enligt kap. 6.1.

7 UTVECKLING AV NYA TILLÄMPNINGAR AV SIP-SYSTEMET

7.1 Inledning

Antag att applikationen HILBERT skall konstrueras. Nödvändiga moduler, som måste skapas är: (små bokstäver är valfria, stora är namnstandard).

hilbert.FOR	Innehåller själva applikationen i Fortran-kod.
GLBhlb.INC	Deklaration av variabler som skall vara globala i applikationsprogrammet. Ej nödvändig om globala variabler saknas.
INIT.DAT	Ger startvärden.
simulant.FOR	Innehåller simuleringsprogram. Ej nödvändigt om man utnyttjar simulering av typ 1 kap. 6.3.
opcscr.DAT	Instruktioner för hur displayen skall utformas. Namnet anges i INIT.DAT.
START.COM	Kommandofil för VAX som anger bl.a. vilka processer som skall startas, och i vilken ordning, Innehåller ev. meny.
START.CMD	Samma kommandofil som START.COM, fast skriven i PDP-datorns kommandospråk.

Dessa filer bör läggas i en egen katalog (directory), som är default-directory för en användaridentitet med samma namn och password som applikationen. Detta för att operatören skall logga in på detta sätt (se kap. 6.1). Katalogerna bör ha samma namn i VAX och PDP, dvs VAX:en anpassas till PDP:ns namnstandard (numeriska directory). Detta förutsättes vid automatisk överflyttning av hela systemet mellan datorerna via floppy disk, vilket sker på följande sätt:

```
1 Logga in på VAX och flytta till $SIP$-katalogen.

2 Placera flyttnings-disk i övre floppy-disk-drivern på VAX.

3 Flytta från VAX till floppy-disk:
  $ @$SIP$:VAXRX2.COM
  OBS. Flera skivor behövs.

4 Flytta skivan till PDP:n och logga in. Account=SIP Password=SIP.

5 Skriv kommandot "MOUNT DY0 DY0" på PDP:n.

6 Flytta från floppy-disk till PDP med kommando "@DY0:A300,310ARX2PDP.CMD"

7 Skriv kommandot "DISMOUNT DY0" på PDP:n.

8 Stoppa in nästa floppy-disk i PDP.

9 Skriv kommandot "MOUNT DY0 DY0" på PDP:n.

10 Flytta från floppy-disk till PDP med kommando "@A300,310ARX2PDP.CMD"

11 Skriv kommandot "DISMOUNT DY0" på PDP:n.

12 Bygg hela systemet med kommando "@A300,310ATOTBYG.CMD".

13 Logga ut från PDP med kommandot "LOGOUT"

Punkt 8-11 kan behövas upprepas flera gånger då inte hela systemet får plats
på en floppy-disk.
```

I VAX:en kan man dock utnyttja möjligheten med logiska namn, för att kunna referera till katalogen med applikationens eget namn (t.ex. $HILBERT$). Detta namn används också som kommando i SIP-systemet, för flyttning mellan olika kataloger, se manualen för VAX/VMS och kap. 5.1.

För att skapa en ny användaridentitet med tillhörande huvudkatalog, vilket man bör göra vid utveckling av en ny tillämpning, krävs systemprivilegier. Detta ligger därför utanför den normala konstruktörens befogenhet, varför detta måste ordnas av någon som har erforderliga privilegier.
Det går naturligtvis bra att köra två eller flera tillämpningar från samma användaridentitet och katalog, men detta är olämpligt, då man går miste om ett flertal av SIP-systemets fördelar och säkerhetsåtgärder mot mänskliga misstag.

Konton avsedda för utveckling av nya tillämpningar, finns installerade i både VAX- och PDP-datorn på TFL, med användaridentiteten SIP och lösenordet SIP.

7.2 Kommandofiler

Kommandofiler är filer uppbyggda av instruktioner skrivna i datorns eget kommandospråk (dvs. operativsystemets språk) och blir därför olika på VAX och PDP:

Vid skapandet av användaridentiteter för varje applikation skall man (som alltid vid skapande av identiteter i de aktuella operativsystemen) skriva en loginfil. Dessa är i SIP-systemet standardiserade på följande sätt:

	VAX	PDP
NAMN:	LOGIN.COM	LOGIN.CMD
KOD:	$ @START $ LOGOUT/FULL	@START LOGOUT

Filerna START.COM och START.CMD ger konstruktören en stor frihet att utforma varje applikation. Man kan t.ex. förse filerna med ett system med menyer för val av olika varianter av applikationen (se exempel kap. 8). Funktioner som måste ingå i STARTFILEN är de som startar och stannar de processer som skall användas vid applikationen.

De som måste användas för att få ett fungerande system är:

INIT (ej process, utan "vanligt" program)

CLOCK

RTPHWE eller någon form av simulering

En eller flera applikationsprocesser

OPCOM

Övriga delar av systemet kan pusslas mera fritt. En begränsning i friheten är PDP-datorns primärminnesutrymme, vilket medför att systemet ej kan göras för komplext, då endast ett fåtal processer (delvis beroende på deras storlek) får plats. Detta kan åtgärdas genom att installera SIP-systemet på en mera kraftfull dator.

DSKOUT måste startas om lagring av mätdata skall kunna ske. Den första typen av simulering enligt kapitel 6.3 sker genom att RTPSWE och DSKINP startas i stället för RTPHWE. Simulering enligt den andra metoden i kapitel 6.3 sker genom att simuleringsprogrammet startar i stället för RTPHWE.
Start sker genom att anropa automatgenererade kommandofiler av typen STV i VAX och STP i PDP. Stopp sker med kommandon enligt exempel.

De olika processerna i SIP-systemet måste vid körning i PDP installeras som "tasks", se kapitel 2.4. Detta sker automatiskt när PDP-datorns operativsystem startas, genom att kommandofilen SIP:INSPRG.CMD (se kapitel 5.6) kompletteras med raderna:

```
.ifins HILBERT rem HILBERT
ins Ä300,325ÅHILBERT
```

Exempel (mall) på START-filer i VAX och PDP:

VAX

```
$!KOMMENTAR: STARTFIL I VAX
$ ON CONTROL_Y THEN GOTO END
$ ON ERROR THEN GOTO END
$ SET NOON
$!
$ SET DEFAULT Ä300,325A
$!
$ RUN Ä300,310AINIT
$!
$ @Ä300,310ACLOCK.STV
$!
$ @Ä300,310ASIMULERING.STV
$ @Ä300,325AHILBERT.STV
$!
$ RUN Ä300,310AOPCOM
$!
$ STOP XHILBERT
$ STOP XSIMULERING
$!
$ STOP XCLOCK
$!
$ END:
```

PDP

```
;KOMMENTAR: STARTFIL I PDP
SET TERM /FULL/NOWRAP/FORM
;
;
;
SET DEFAULT Ä300,325A
;
RUN Ä300,310AINIT
;
@Ä300,310ACLOCK.STP
@Ä300,310ADSKOUT.STP
@Ä300,310ARTPHWE.STP
@Ä300,325AHILBERT.STP
;
RUN Ä300,310AOPCOM
;
ABORT/TASK HILBERT
ABORT/TASK RTPHWE
ABORT/TASK DSKOUT
ABORT/TASK CLOCK
```

Observera att OPCOM körs som den process som användaren själv utnyttjar i operativsystemet (ej installerad som task). Eftersom det är en föredel under utveckling av en ny applikation att kunna köra denna från olika användaridentiteter utan att bli utloggad efteråt, så har startfilens kod ej skrivits direkt i login-filen.

Vissa kommandon i PDP börjar med en punkt (.). Programrader med en . som första tecken misstolkas vid dokumentationsgenerering i VAX. Dessa rader får därför ej medtagas vid dokumentation. Använd %DOCEND% enligt exempel i kapitel 7.5 och 8.7.

7.3 Initieringsfil

När systemet startas, körs programmet INIT, som läser filen INIT.DAT från aktuell katalog.

Där finns följande uppgifter:

Kommando: OPCPIN nr s9999.99. kommentar
sätter default-värdet
s9999.99 i cellen benämnd OPCRIN (nr).

Kommando: LOGG nr kommentar

Markerar att cell nr nr i IOAREA, se kap. 4.3 skall lagras vid lagring.

Kommando: NAMN nr namn

Placerar textsträngen namn (30 tecken) i cellen OPCCMN (nr).

Följande celler är reserverade:

OPCCMN(1): Filnamn där data om displayens utseende finns. (se kap. 7.4)

OPCCMN(2): Namn på fil där data skall lagras i binär form

OPCCMN(3): Namn på fil där data skall lagras i ASCII-kod.

Kommando: TID nr 9999 kommentar.

Sätter samplingstiden för processen associerad med eventflag nr nr. Nr är här ett tal mellan 65 och 75 där följande nummer är reserverade:

65: RTPHWE, eller RTPSWE

66: DSKINP, eller DSKOUT

67: Applikation

68: OPCOM

69: Simuleringsprogram.

Övriga är till för eventuell utbyggnad av systemet, t.ex. flera parallella applikations- och simuleringsprogram.

Enheten är 2 tick = 40 ms = LDELTA i OPCOM.

Observera att processerna under körning kan ställa om samplingstiden genom att ändra värdet på variabeln HWEWTM(nr). Initieringfilen ger bara ett defaultvärde på HWEWTM(nr).

Exempel på initieringsfil:

```
OPCPIN 01 +0000.00  Defaultvärde för BÖRVÄRDE
OPCPIN 02 +0003.00  Defaultvärde för VARIANT
LOGG   01
LOGG   17
NAMN   01 Ä300,312AOPCSCR.DAT
NAMN   02 Ä300,312ALOG.BIN
NAMN   03 Ä300,312ALOG.ASC
TID    65 0002
TID    66 0002
TID    67 0002
TID    68 0030
TID    69 0002
```

7.4 Design av display

I filen INIT.DAT anges med hjälp av NAMN 1 namnet på den fil, där man lagrat instruktioner för hur displayen skall se ut. Följande instruktioner finns, där x anger rad nr och y anger kolumn nr på skärmen där angiven parameter skall skrivas ut:

OPCRIN nr x y kommentar

Data för inläsning till cellen OPCRIN(nr) placeras i pos x, y med formatet F 9.2.

OPCPIN nr s9999.99 kommentar

Ger cellen OPCRIN(nr) defaultvärdet s9999.99 (s=tecken)

OPCRUT nr x y kommentar

Utskrift av värdet i cellen OPCRUT(nr) skrivs ut i pos. x, y med format F 9.2

OPCIUT nr x y typ "text".

Om "typ" är TAL sker utskrift av värdet i OPCIUT(nr) med format I 6.2.

Om "typ" är LOG visas "text" vid pos x, y om OPCIUT(nr)≠0. Om OPCIUT(nr)=0 skrives "text" över med blanktecken. Detta kan användas för att indikera händelser på skärmen. Om "typ" är TXT gäller samma som för "typ"=LOG, med undantag för när OPCIUT(nr)=0. Då skrivs inga blanktecken ut på skärmen.

TEXT nr x y "text"

Textsträngen "text" skrives ut på skärmen i pos x,y. Nr är en löpande numrering från 1 till 50.

GRFDEF X_1 Y_1 X_2 Y_2 kommentar

Definierar ett grafiskt fönster där X_1Y_1 anger övre vänstra hörnet, X_2Y_2 anger nedre högra hörnet. Endast ett grafiskt fönster kan vara definierat samtidigt. Dessa koordinater motsvarar ej de koordinater som anges i det alfanumeriska fönstret. Hela skärmen motsvarar GRFDEF 01 01 24 80.

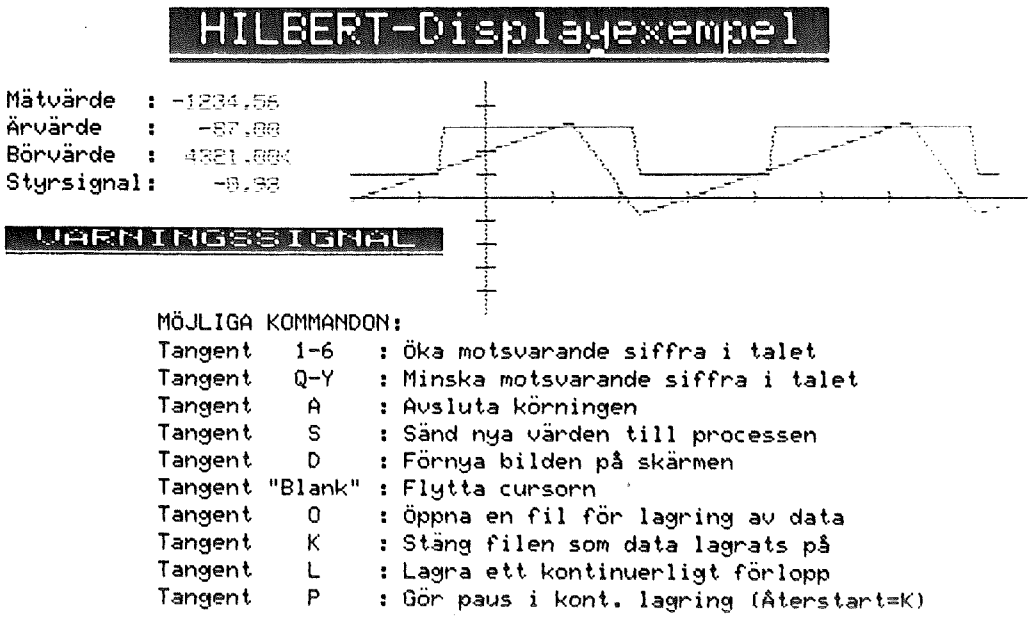
OPCGUT nr kommentar

Medför att värdet av OPCGUT(nr) kommer att ritas ut i det grafiska fönstret. Nr kan vara 01, 02, 03, 04.

Exempel på en datafil:

```
TEXT   01 02 09 <ESC>#3<ESC>Ä1;4;7m HILBERT-Displayexempel <ESC>ÄOm
TEXT   02 03 09 <ESC>#4<ESC>Ä1;4;7m HILBERT-Displayexempel <ESC>ÄOm
TEXT   03 01 01 <ESC>Ä1m
TEXT   04 05 05 Mätvärde  :
OPCRUT 01 05 16 Mätvärde
TEXT   05 06 05 Ärvärde   :
OPCRUT 02 06 16 Ärvärde
TEXT   06 07 05 Börvärde  :
OPCRIN 01 07 16 Börvärde
TEXT   20 08 05 Styrsignal:
OPCRIN 02 08 16 Styrsignal
OPCIUT 01 10 03 LOG <ESC>#6<ESC>Ä1;4;7m VARNINGSSIGNAL <ESC>ÄOm
TEXT   07 13 16 MöJLIGA KOMMANDON:
TEXT   08 14 16 Tangent   1-6   : öka motsvarande siffra i talet
TEXT   09 15 16 Tangent   Q-Y   : Minska motsvarande siffra i talet
TEXT   10 16 16 Tangent    A    : Avsluta körningen
TEXT   11 17 16 Tangent    S    : Sänd nya värden till processen
TEXT   12 18 16 Tangent    D    : Förnya bilden på skärmen
TEXT   13 19 16 Tangent "Blank" : Flytta cursorn
TEXT   14 20 16 Tangent    O    : öppna en fil för lagring av data
TEXT   15 21 16 Tangent    K    : Stäng filen som data lagrats på
TEXT   16 22 16 Tangent    L    : Lagra ett kontinuerligt förlopp
TEXT   17 23 16 Tangent    P    : Gör paus i kont. lagring (Aterstart=K)
TEXT   49 01 01 <ESC>ÄOm
GRFSCR 07 30 12 80
OPCGUT 01 Kurva 1 är aktiverad
OPCGUT 02 Kurva 2 är aktiverad
OPCPIN 01 +4321.00  Defaultvärde för börvärde
OPCPIN 02 -0000.98  Defaultvärde för styrsignal
```

Denna ger följande display vid körning:



Figur 7.1 Display producerad enligt instrutionerna i exemplet på datafil.

Utskrifter i färg, fet stil och/eller inverterad skärm (ofärgad skrift på färgad bakgrund) styrs med escape- (<ESC>) sekvenserna vid utskrift av text, se vidare manualen för Textronix-terminalen. Textutskrifterna sker i den ordning som numreringen i indatafilen ger, vilket måste beaktas vid omställning av terminalen med dessa escape-sekvenser.

7.5 Källkodsfiler

Med källkodsfiler menas här i första hand, de programmoduler i systemet, som tidigare benämnts applikation. Den mall som här redovisas är dock gällande för alla processmoduler i SIP-systemet, även de som benämns som "fasta", och eventuellt simuleringsprogram.

De fortran-kommandon som ingår i det följande exemplet, är stavade antingen med stor eller liten bokstav. Detta för att skilja de kommandon, som kan betraktas som "obligatoriska" i en processmodul, vilka stavas med stor bokstav, från de som endast utgör exempel på tänkbart kommando i den aktuella applikationen, vilka stavas med liten bokstav.

Antag att programkoden för processen Hilbert skall skrivas. I processen behöver man "minnas" vissa variabler i under-programmen mellan två anrop (vanlig situation vid digital reglering). Detta löses enklast genom införandet av process-globala variabler, vilka enligt tidigare resonemang bör de-klareras i en separat modul, som enligt namnstandarden får namnet GLBHIL.

```
c       Detta är modulen GLBHIL.INC, med deklaration av
c       processglobala variabler för applikationen Hilbert.

c       Gamla mätvärden som skall "minnas" :
        real*4 gold1,gold2
c       gold1 = mätvärde 1 från föregående sampling
c       gold2 = mätvärde 2 från föregående sampling

c       Gamla värden på flaggvektor:
        integer*2 gminne(1:5)
c       gminne = vektor bestående av fem flaggor


c       DEKLARATION AV COMMONBLOCK:
        COMMON /GLBHIL/
        1
        1       gold1,
        1       gold2,
        1       gminne
```

I programkoden för huvudprogrammet införes referenser %DOCNOD% .. enligt exemplet vilka medför att denna tillämpning inordnas i SMS-trädet. För att detta skall fungera måste även en referens i filen SIP:SIP.TXT anges, som motsvarar den i källkodsfilen. Man skriver t.ex. %DOCNOD% PRC $HILBERT$:HILBERT.FOR. Detta medför att automatgenererade filer skapas.

För att även automatisk kompilering och länkning skall ske, måste filerna SIP:TOTBYG.COM och SIP:TOTBYG:CMD, kompletteras med anrop av de automatgenererade kommando-filerna av typen CMV respektive CMP. Till exempel $ É<300,325>HILBERT.CMV och É<300,325>HILBERT.CMP.

Programexempel med kommentarer till den skrivna koden (d.v.s. kommentarer även till programkodens kommentarer) inom parentes:

```
c IDENT A300,325AHILBERT.FOR
c (Kommentar som talar om filens namn)

      PROGRAM HILBERT

c*************************************************************
c
c       Detta är applikationen HILBERT. Den används
c       som exempel på applikation i TFL-rapporten:
c       SIP-SYSTEMET, Ett datorsystem för reglertekniska
c       experiment och mätdatabehandling.
c
c*************************************************************

c--------TABELL ÖVER ANSLUTNINGAR TILL PROCESSINTERFACE---

c       I3644(0)=Mät-värde
c       I3644(1)=Är-värde
c       O5531(0)=Styr-signal

c--------TABELL ÖVER ANSLUTNINGAR TILL OPCOM---------------

c       OPCIUT(1)=Varnings-signal
c       OPCRUT(1)=Mät-värde
c       OPCRUT(2)=Är-värde
c       OPCRUT(3)=Styr-signal
c       OPCRIN(1)=Bör-värde

c-------------------------------------------------------------

c       %DOCNOD% LBR $SIP$:LBRXXX.FOR
c       (Anger kopplingen till biblioteket LBRXXX.)

c       %DOCNOD% CON $SIP$:CMNHWE.INC
        INCLUDE 'A300,310ACMNHWE.INC'
c       (Anger kopplingen till CMNHWE.)

c       %DOCNOD% CON $SIP$:CMNOPC.INC
        INCLUDE 'A300,310ACMNOPC.INC'
c       (Anger kopplingen till CMNOPC.)

c       %DOCNOD% GLB $HILBERT$:GLBHIL.INC
        INCLUDE 'A300,325AGLBHIL.INC'
c       (Anger kopplingen till GLBHIL.)

c       %DOCNOD% PRI $SIP$:DUMMY.DUM PRI=25
c       (Anger att processens prioritet skall vara 25.)

c       Filer som skall medtagas som underkapitel i
c       den automatgenererade dokumentationen:
c       %DOCNOD% DOC $HILBERT$:GLBHIL.INC (ev. titel)
c       %DOCNOD% DOC $HILBERT$:START.COM (ev. titel)
c       %DOCNOD% DOC $HILBERT$:START.CMD (ev. titel)

c       %DOCEND%
c       (Markerar slutet på den löpande text som skall
c       medtagas vid automatgenererad dokumentation
c       i filen $SIP$:SIP.LIZ)
```

```
c=========START HUVUDPROGRAM=========================== %DOC%
c       (%DOC% = Enstaka kommentarrad som medtages
c       i dokumentation.)

c       (Deklaration av lokala variabler)

c       integer*2 Lstat
c       Lstat=Statusflagga vid anrop av LBRXXX-rutiner

c-------------------------------------------------------------

        print *,'HILBERT har startat'

c       (Rader som ger startvärden åt variablerna, m.m.)

c       Initiera globala eventflags
        CALL XCRGEF(Lstat)

c---------START LOOP-----------------------------------------

1000    CONTINUE

c       (Rader som anger vad som skall utföras vid
c       varje sampling.)

        CALL    XCLREF(67,Lstat)
        CALL    XWEF(67,Lstat)
c       (Dessa två satser gör att processen står i
c       väntetäge ett samplingsintervall, dvs.
c       väntar på att CLOCK skall sätta eventflag nr 67.

        GOTO 1000
c       (Låt processen gå i en oändlig loop.)

c---------END LOOP-------------------------------------------

c       (Eventuella åtgärder efter att loopen är bruten.)

        END

c=========END HUVUDPROGRAM============================ %DOC%

c       %DOCBEG%
c       (Återuppta kontinuerligt inkluderande
c       kommentarrader i dokumentation.)

        subroutine sub1(Pvalue,.....)

c       %DOCNOD% GLB $HILBERT$:GLBHIL.INC
        INCLUDE 'X300,325AGLBHIL.INC'
c       (Anger kopplingen till GLBHIL.)

c       real*4 Lvalue

c       (Deklaration av underprogram)

c=======end sub1=======================================

c       %DOCEND%
```

Applikationsprocessen kommunicerar med omvärlden via följande tidigare omnämnda variabler. Här bör man notera variabelns typ och definitionsområde:

I3643(0:15),I3644(0:7)

Typ: heltalsvariabler = integer.
Definitionsmängd: [-2048, 2047]
Representerar: Analoga indata från RTP-interface

O5531(0:3),O5522(0:0)

Typ: heltalsvariabler = integer
Definitionsmängd: [-2048, 2047]
Representerar: Analoga utdata till RTP-interface.

I3820(0:0),I3582(0:0),I3737(0:0)

Typ: heltalsvariabler = integer
Definitionsmängd: [-32768, 32767]
Representerar: Digitala insignaler från RTP interface, det decimaltal som utgör värdet av 16 binära siffror (= digitalt ord).

O3820(0:0),O3581(0:0)

Typ: heltalsvariabler = integer

Definitionsmängd: [-32768, 32767]

Representerar: Digitala utsignaler till RTP-interface.

OPCRIN(1:OPCRIA)

Typ: reelt värde = real

Definitionsmängd: [-9999,99, 9999,99]

Representerar: Värde given av operatören från terminal.
Obs! högst två decimaler.

OPCRUT(1:OPCRUA)

Typ: reellt värde = real

Definitionsmängd: [-9999,99, 9999,99]

Representerar: Värden från applikationsprogram som skall skrivas ut på skärmen.

OPCIUT(1:OPCIUA)

Typ: heltalsvariabel = integer

Definitionsmängd: [-9999,9999]

Representerar: Värdena från applikationsprogram som skall skrivas ut som heltal på skärmen, eller

händelseindikering (händelse = OPCIUT(X)≠0). Vilket som gäller anges i datafil enligt kapitel 7.4.

OPCGUT(1:4)

Typ: heltalsvariabel = integer

Definitionsmängd: [-1000, 1000]

Representerar: Värden från applikationsprogram som skall skrivas ut grafiskt på skrämen. Ett skalstreck på skärmen motsvarar 200 skalenheter i OPCGUT.

OPCKEY

Typ: heltalsvariabel = integer

Definitionsmängd: [0, 127]

Representerar: ASCII-koden för senaste tangentnedtryckning på terminalen, som OPCOM har registrerat.

Det är lämpligt att man i applikationsprogrammet kontrollerar att erhållna värden håller sig inom resp. definitionsmängd. Annars kan mycket svårupptäckta exekveringsfel uppstå.

Ett närbesläktat exekveringsfel, som ofta orsakas av underlåtenhet att kontrollera indata, är att ett heltalsvärde hamnar utanför datorns definitionsområde för heltal, när man omvandlar ett tal från reelt till heltal.

Detta fel, kallat integer overflow, kan naturligtvis uppstå

även om ovanstående kontroll är utförd genom uträkningar i programmet.

Därför rekommenderas en kontroll av att det reella värdet, ej är för stort, före omvandlingen, då exekveringsfel av denna sort medför att processen stannar.
Ett underprogram i fortran som utför omvandling (avrundning) från reellt tal till heltal med denna kontroll, ser ut på följande sätt:

```
c=============================================================
      subroutine RIOMV(PINT,PREAL,PSTAT)

c     Denna rutin ger avrundning av det rella
c     talet PREAL till närmaste heltal, PINT.
c     Vid integer overflow fås värdet 0. PSTAT får
c     värdet 1=för stort tal, -1=för litet, 0=lagom.

      real*4 PREAL
      integer*2 PSTAT,PINT

c-------------------------------------------------------------

      if (PREAL.ge.32767.0) then
          PINT=0
          PSTAT=1
      elseif (PREAL.le.-32768.0) then
          PINT=0
          PSTAT=-1
      else
          PINT=nint(PREAL)
          PSTAT=0
      endif

      return
      end

c=======end RIOMV=============================================
```

7.6 Simuleringsprogram

Vid framställning av simuleringsprogram göres ett nytt applikationsprogram enligt samma mall som i kap. 7.5, som placeras i samma katalog, med följande ändringar i mallen:

* Byt namnet (Hilbert) på alla förekommande ställen
* Stryk kopplingen till OPCOM
* Ändra prioritet (två processer bör ej ha samma prioritet)
* Gör ny eventuell Include-fil för processglobala variabler
* Eventflag nr 67 bytes ut mot eventflag nr 69. (se kap 7.3)

Simuleringsprogrammet skall läsa ur de minnesceller i IOAREA som börjar på O och skriva i de som börjar på I (d.v.s. tvärt emot applikationen). Observera att simuleringsprogrammet måste inordnas i SMS-systemet på samma sätt som applikationsprogrammet.

7.7 Automatgenererade filer

När filerna enligt uppräkningen i kap. 7.1 är skrivna, används SMS-systemet för att skapa alla övriga filer som ingår i ett fungerande system. Lämpliga kommandon är följande:

$ ÉSIP:TOTALBYGG.COM (endast i VAX)

Bygger en ny version av SIP-systemets alla filer, inkl. dokumentation, kompilering och länkning (= tar lång tid att exekvera). Nödvändig vid skapande av ny tillämpning.

$ ÉSIP:TOTBYG.COM (VAX) och É<300,310>TOTBYG.CMD (PDP)

Kompilerar och länkar hela SIP-systemet, tar lång tid, speciellt i PDP.

$ É$HILBERT$:HILBERT.CMV (VAX), É<300,325>HILBERT.CMP (PDP)

Kompilerar och länkar tillämpningen. HILBERT (med automatgenererade kommandofiler).

$ SMS/PERFORM SIP:HILBERT.FOR STV (endast i VAX)

Skapar den automatgenererade filen HILBERT.STV.

Vilka de automatgenererade filerna, som skapas av TOTALBYGG.COM, är framgår av kap. 5.5
Speciellt bör dock noteras filerna av typerna *.STV, och *.STP, vilka ombesörjer start av processer (tasks), se kapitel 7.2.

8 LINDA, EXEMPEL PÅ EN REGLERTILLÄMPNING AV SIP

8.1 Inledning

I detta kapitel skall en digital PID-regulator för att reglera försöksuppställningen i kapitel 2.3, enligt schemat i kapitel 7, implementeras.

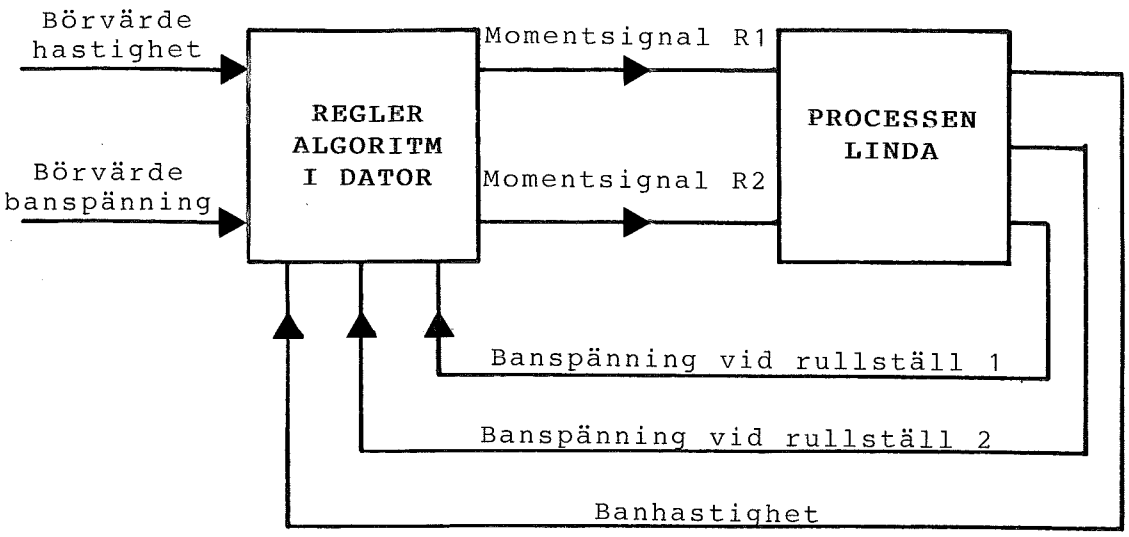
Applikationen döpes till LINDA och inplaceras i SIP-systemet i katalogen <300,311> (d.v.s. applikation nr 1).

För utveckling av programmet LINDA.FOR utnyttjas ett simuleringsprogram kallat LINSIM.FOR.

8.2 Processmodell och regleralgoritm

Det första steget i konstruktionen av en regulator är att bilda sig en uppfattning om strukturen hos processen som skall regleras, och att försöka beskriva den schematiskt, ett försök till en sådan modell göres i Figur 8.1.

Efter att man bestämt processens struktur kan man söka matematiska samband mellan signalerna, för att kunna bestämma en regleralgoritm.

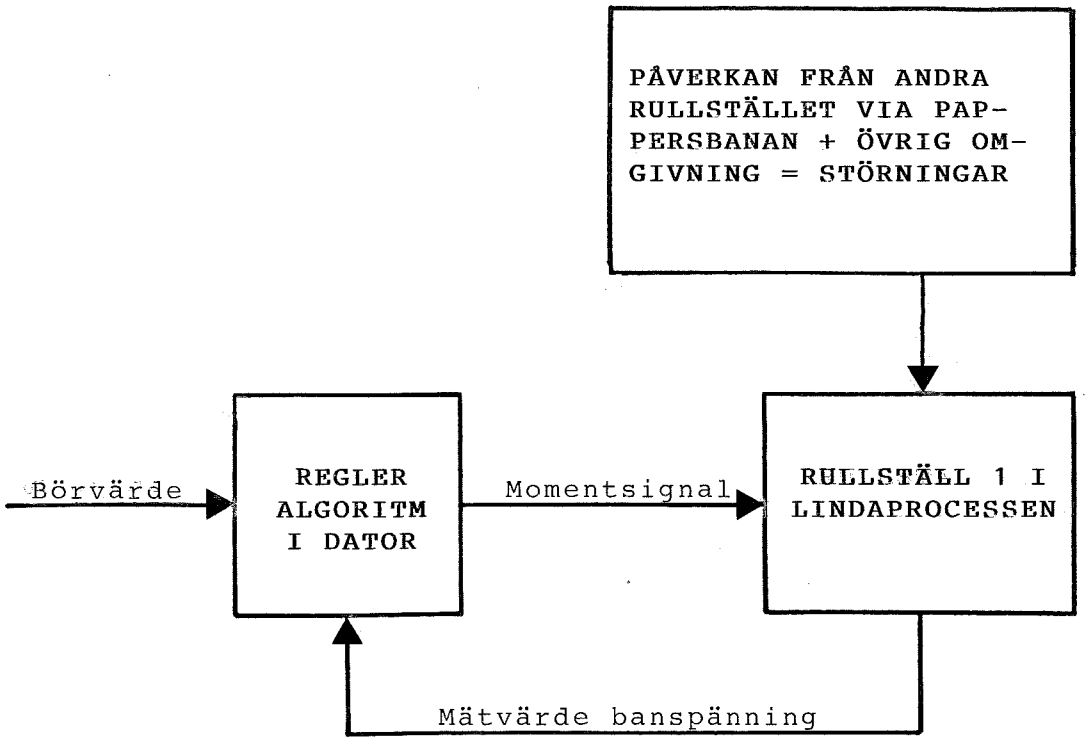


Figur 8.1 Schematisk bild av processen Linda med datorstyrning

En modell av den typ som är skisserad i figur 3.1, leder dock till mycket komplicerade samband, varför man bör splittra modellen i delsystem, där man kan använda känd reglerteori och i bästa fall rena standardlösningar på problemet. De faktorer som påverkar processen och som inte omfattas av en förenklad modell betraktas som störningar, vilka är små i förhållande till de faktorer som medtages i modellen, se Figur 8.2.

I detta exempel utnyttjas regulatorn från kapitel 3.3 som regleralgoritm i programmet, vilket innebär en digital "översättning" av den befintliga regulatorn i systemet, se kapitel 2.3.

Experiment med olika mer komplicerade algoritmer underlättas genom att låta själva regleralgoritmen utgöra ett underprogram, vilket med lätthet kan bytas ut utan att göra några större ingrepp i resten av mjukvaran.



Figur 8.2 En förenklad modell av Linda uppbyggd av delsystem.

8.3 Frekvensanalys

För att kunna avgöra valet av samplingstid bör man utföra en frekvensanalys av de signaler man tänker använda. (Man kan även frekvensanalysera utgående signaler för att kontrollera att de ser korrekta ut).
Ointressanta frekvenser bör filtreras bort före A/D-omvandling i process-interfacet.

Intressanta frekvenser i det här sammanhanget ligger i området 0-ca 5 Hz. Då de inbyggda anti-aliasingfiltren har en brytfrekvens på 11 Hz (enligt försök är dessutom frekvens-

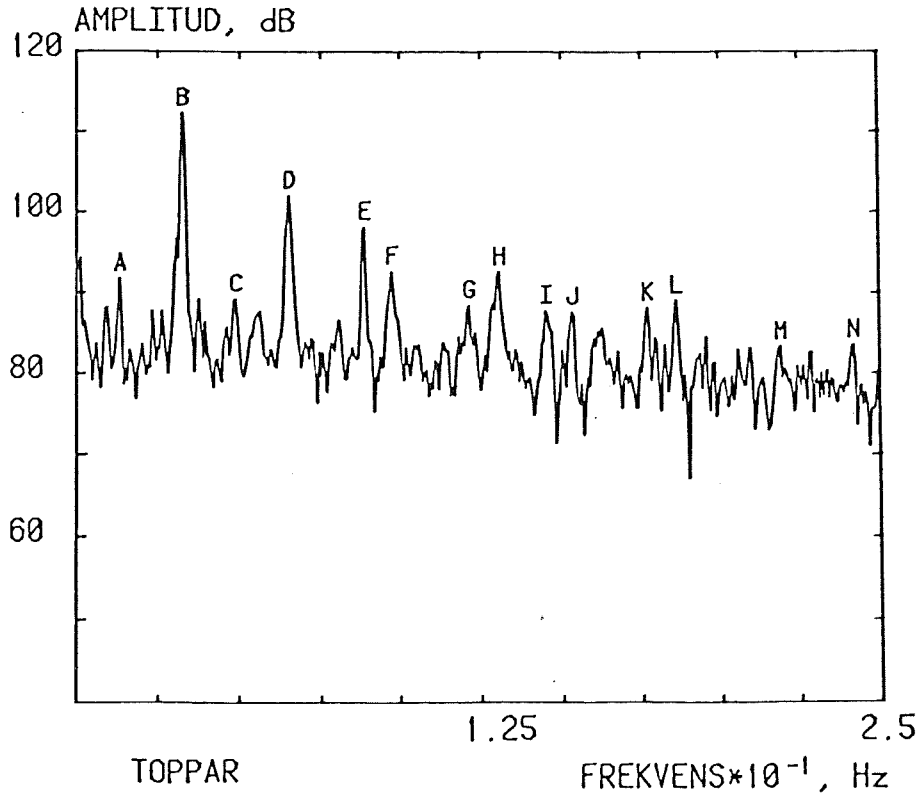
kurvan efter filtrering ganska flack) ger samplingsteoremet [4] att man med de "kritiska" delarna av SIP-systemet bör sampla ungefär med tiden 2 tick (25 Hz), vilket hade kontaterats vara snabbast praktiskt genomförbara samplingstid. Efter samplingen kan man bygga in digitala filter för att bli av med oönskade frekvenser. Detta är väsenligt då olika delar av systemet kommer att sampla olika fort och att man därigenom kan råka ut för aliasing-problem vid den interna kommunikationen. Detta är inte utfört i den här redovisade versionen av SIP-systemet, då man med kännedom om den reglerade processens uppförande kan konstatera att sannolikheten för att detta inträffar är mycket liten.

En frekvensanalys av signalerna från processen LINDA visar att där förekommer en resonanstopp vid ca 3 Hz, vilken om den ej går att reglera bort med någon reglerstrategi, bör filtreras bort.

Vid olika typer av försök förekommer samma frekvens i systemet, varför man kan dra slutsatsen att de uppkommer som mekaniska svängningar i rullställen (motor-transmission-rulle).

Vid regleringen måste man också tänka på att när regleralgoritmen utgörs av en diskret approximation av en kontinuerlig algoritm, måste samplingstiden vara tillräckligt snabb för att vara försumbar jämfört med tidskonstanter i systemet.

Om man är hänvisad till att utnyttja längre samplingstider, bör man övergå till algoritmer, härledda enligt teorin för digitala system.



KOD	FREKVENS	AMPLITUD dB	AMPLITUD V RMS
A	1.313	91.8	3.89E-002
B	3.250	112.3	4.12E-001
C	4.875	89.2	2.88E-002
D	6.500	102.0	1.26E-001
E	8.875	98.1	8.04E-002
F	9.750	92.6	4.27E-002
G	12.125	88.4	2.63E-002
H	13.063	92.5	4.22E-002
I	14.500	87.7	2.43E-002
J	15.313	87.5	2.37E-002
K	17.688	88.1	2.54E-002
L	18.625	89.0	2.82E-002
M	21.875	83.3	1.46E-002
N	24.188	83.5	1.50E-002

Figur 8.3 Frekvensspektrum för processen LINDA.

8.4 Driftsfall

För att Linda skall kunna användas till olika experiment, krävs att man skall kunna reglera med både banspänning och hastighet som börvärde på bägge rullställen. Möjlighet att styra ut en konstant momentsignal, t.ex. momentet 0, är också önskvärt. Observera att styrsignalen är en annan

storhet (dvs. moment) än är- och börvärden vid reglering. Alla storheterna representeras dock av signaler mellan 0 och 10 V, varför en normal regleralgoritm kan användas, med lämpligt val av parametrar (vilka ej blir dimensionslösa). De tänkbara driftsfallen är följande:

1 Banspänningsreglering rulle 1/Hastighetsreglering rulle 2
2 Banspänningsreglering rulle 2/Hastighetsreglering rulle 1
3 Banspänning/Banspänningsreglering
4 Manuell momentstyrning
5 Manuellt moment rulle 1/Banspänningsreglering rulle 2
6 Manuellt moment rulle 2/Banspänningsreglering rulle 1
7 Manuellt moment rulle 1/Hastighetsreglering rulle 2
8 Manuellt moment rulle 2/Hastighetsreglering rulle 1
9 Styrning av börvärden Banspänning/Hastighetsreglering
10 Styrning av börvärden Banspänning/Banspänningsreglering

Vid applikationen LINDA kompletteras denna lista med driftsfallet "Datorstyrning avstängd"(nummer 0), då man har möjlighet att erhålla alla önskade mätdata på skärmen, samt lagra mätdata i datafil, samtidigt som styrningen helt sker med den analoga utrustningen.

Dessa olika driftsfall utgör i sig olika applikationer av SIP-systemet, men då skillnaden är liten mellan dem, köres de med samma reglerprogram, där CMNOPC-cellen OPCRIN(1) innehåller information till LINDA-modulen vilket driftsfall som skall exekveras.

Detta innebär att driftsfallen kan ändras under pågående körning utan att avbryta någon pågående aktivitet.

Önskvärt är också att ha olika display-varianter för de olika driftsfallen. Detta löses på ett smidigt sätt, genom att med kommandot A (a) endast avsluta körning av OPCOM (styrs med kommandofil enligt kapitel 8.5). Reglerprocessen fortsätter då att reglera med senast inställda värden, medan operatören via en meny, kan välja en ny variant av display och sedan återstarta OPCOM. I samband med inläsning av nya display-instruktioner, erhålles ett nytt default-värde i cellen OPCRIN(1), vilken inte är åtkomlig för ändring på annat sätt i denna applikation, och därigenom får reglerprocessen information om eventuellt nytt driftsfall (reglermod).

8.5 Design av display

Möjligheten att välja olika driftsfall med egna unika display-utformningar medger också möjlighet att välja olika alternativa displayer för varje driftsfall. Då grafiken är en tidsödande process, skulle man vilja välja bort den i vissa fall. För att ytterligare öka valmöjligheterna förses Linda med 4 alternativa displayer:

1 Numerisk presentation av alla mätvärden
2 Grafisk presentation av är- och börvärden för båda rullställen
3 Grafisk presentation av är- och börvärden för rullställ 1
4 Grafisk presentation av är- och börvärden för rullställ 2

Utöver denna presentation av mätdata måste även erforderliga indata och varningssignaler beredas plats på skärmen.

Indata är

- 2 st börvärden eller styrvärden beroende på reglermod
- 2 st proportionella reglerkonstanter
- 2 st integratortider
- 2 st derivatatider

Lämpliga varningssignaler;

- Banbrottsindikering
- Mättade momentsignaler för rulle 1 och rulle 2
- Felaktiga indata

För att utföra dessa byten krävs alltså en indatafil för varje display-variant. Dessa göres så snarlika att eventuella ändringar kan göras samtidigt i samtliga filer via en kommando-fil.

Dessa indatafiler döpes enligt följande princip:

SCR[Displaytyp][Reglermod].DAT

Dvs SCRN1.DAT = numerisk display reglermod 1, osv.

Figurerna 8.4 - 8.8 som följer, utgör exempel på hur några av Lindas alternativa displayer ser ut.

LINDA, DIGITAL REGLERING

DATORSTYRNING AVSTÄNGD

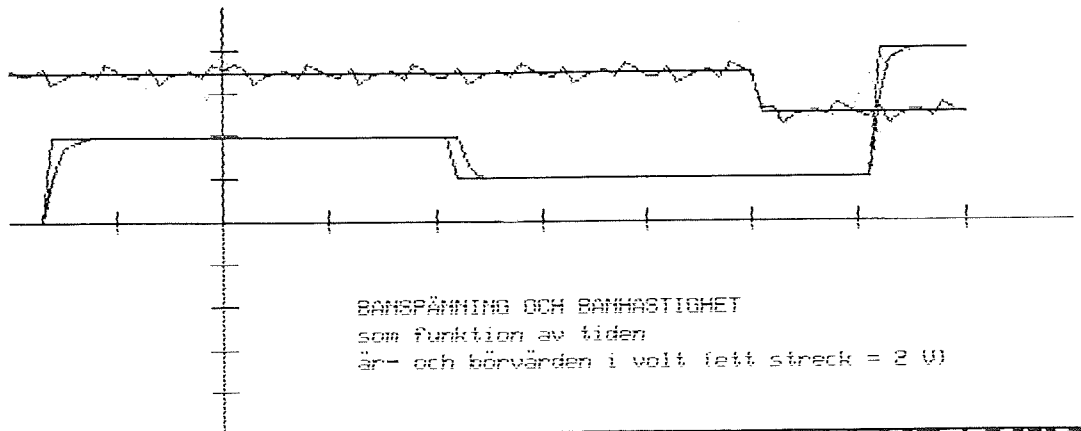
KOMMANDOTANGENTER :

D=Förnya displayen
A=Ändra display eller reglermod
O=Öppna loggfil
K=stäng loggfil
M=Logga ett sample
L=Logga tillsvidare
P=Gör paus i loggning

MÄTVÄRDEN:

Banhastighet börvärde	5.49
Banspänning, rulle 1, börvärde	2.99
Banspänning, rulle 2, börvärde	1.50
Banhastighet ärvärde	-0.12
Banspänning, rulle 1, ärvärde	0.00
Banspänning, rulle 2, ärvärde	0.01
Moment, rulle 1, ärvärde	0.00
Moment, rulle 2, ärvärde	0.00
Varvtal, rulle 1, ärvärde	0.01
Varvtal, rulle 2, ärvärde	0.63

Figur 8.4 Reglermod 0 (SCRN0.DAT), finns endast med numerisk presentation.



LINDA,BANSPÄNNING/HASTIGHETS-REGLERING
BANSPÄNNING RULLE 1, HASTIGHET RULLE 2

INSTÄLLDA VÄRDEN :

Banspänning	=>	2.00
Hastighet	=>	5.00

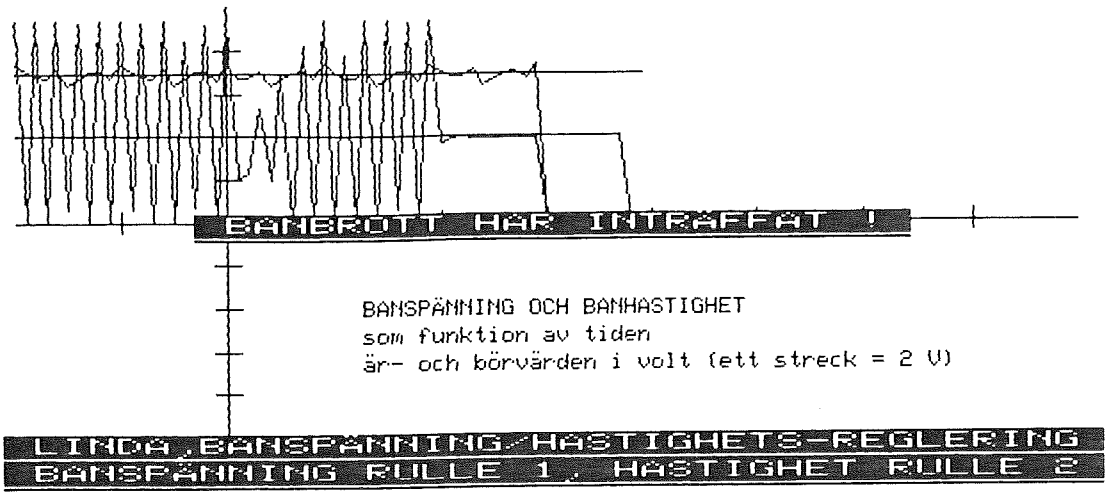
TRYCK TANGENT A (a) FÖR ATT
ÅTERVÄNDA TILL MENYN !

REGULATORKONSTANTER:

Banspänning, P-del	=>	0.15
Hastighet, P-del	=>	0.20
Banspänning, I-del	=>	140.00
Hastighet, I-del	=>	95.00
Banspänning, D-del	=>	25.00
Hastighet, D-del	=>	10.00

Integrator- och derivatatid i ms.
Negativa värden = avstängd I eller D-del.

Figur 8.5 Reglermod 1, grafisk display (SCRG1.DAT).



INSTÄLLDA VÄRDEN :
Banspänning => -28.00<
Hastighet => 6.90
VARNING, ORIMLIG INDATA !
MÄTTAD MOMENTSIGNAL, RULLE 1
MÄTTAD MOMENTSIGNAL, RULLE 2

TRYCK TANGENT A (a) FÖR ATT
ÅTERVÄNDA TILL MENYN !

REGULATORKONSTANTER:
Banspänning, P-del => 0.15
Hastighet, P-del => 0.20
Banspänning, I-del => 140.00
Hastighet, I-del => 95.00
Banspänning, D-del => 25.00
Hastighet, D-del => 10.00
Integrator- och derivatatid i ms.
Negativa värden = avstängd I eller D-del.

Figur 8.6 Samma driftsfall (SCRG1.DAT) som Figur 8.5, men med varningssignalerna aktiverade.

LINDA - DIGITAL REGLERING
BANSPÄNNING RULLE 1
HASTIGHET RULLE 2

VARNING, ORIMLIG INDATA !
INSTÄLLDA VÄRDEN :
Banspänning => 244.00<
Hastighet => 6.90

REGULATORKONSTANTER:

Banspänning, P-del => 0.15
Hastighet, P-del => 1.20
Banspänning, I-del => 30.00
Hastighet, I-del => 95.00
Banspänning, D-del => 15.00
Hastighet, D-del => 10.00

I-tid och D-tid anges i ms.
Negativa värden innebär
avstängd I eller D-del.

MÄTTAD MOMENTSIGNAL RULLE 1

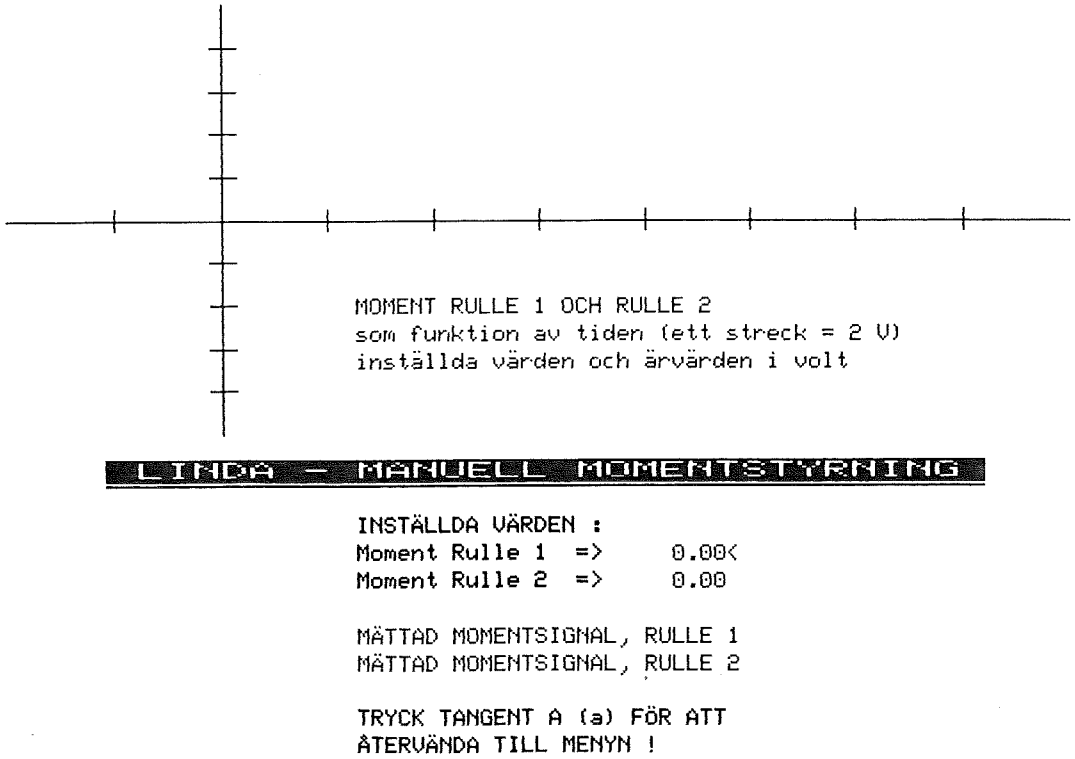
BANBROTT HAR INTRÄFFAT !
TRYCK TANGENT A (a) FÖR ATT
ÅTERVÄNDA TILL MENYN !

MÄTVÄRDEN:

Banhastighet börvärde	5.49
Banspänning, rulle 1, börvärde	2.99
Banspänning, rulle 2, börvärde	1.50
Banhastighet ärvärde	0.50
Banspänning, rulle 1, ärvärde	0.25
Banspänning, rulle 2, ärvärde	0.01
Moment, rulle 1, ärvärde	0.00
Moment, rulle 2, ärvärde	10.21
Varvtal, rulle 1, ärvärde	0.04
Varvtal, rulle 2, ärvärde	0.47

MÄTTAD MOMENTSIGNAL RULLE 2

Figur 8.7 Reglermod 1, med numerisk display (SCRN1.DAT).



Figur 8.8 Reglermod 4, grafisk display (SCRG4.DAT).

Övriga i Linda-applikationen förekommande display-varianter har ett snarlikt utseende.

8.6 Initieringsfil

De värden som anges i initieringsfilen för reglermod kommer att ändras vid körning av meny-programmet, och det som anges här är värdena för avstängd styrning. Detta skall gälla när LINDA startas, vilket sker före menyförfarandet i startfilen. Namnet OPCSCR.DAT motsvarar ingen verklig fil, utan startfilen (se kapitel 8.7) tilldelar den datafil som skall användas namnet OPCSCR.DAT som tillfälligt alternativt namn före start av OPCOM.

För att veta vilka värden som skall lagras, kontrollerar man vilka ingångar som används, och vilka ordningsnummer i IOAREA

dessa motsvarar. Samplingstiderna erhålles som erfarenhetsvärden.

INIT.DAT får följande utformning.

```
OPCPIN 01 +0000.00  REGLERMOD = styrning avstängd
LOGG   05 Aktuella mätvärden:
LOGG   06
LOGG   07
LOGG   08
LOGG   17
LOGG   18
LOGG   19
LOGG   20
LOGG   21
LOGG   22
LOGG   23
NAMN   01 A300,311AOPCSCR.DAT
NAMN   02 A300,311ALOG.BIN
NAMN   03 A300,311ALOG.ASC
TID    65 0002
TID    66 0002
TID    67 0002
TID    68 0030
TID    69 0010
```

8.7 Startfiler med meny

Filerna START.COM och START.CMD är skrivna i operativsystemens egna språk för kommandofiler, och för mer information om deras språkelement, hänvisas till manualerna.

För att samma utskrifter skall erhållas vid körning i VAX och PDP, och att en ändring automatiskt skall gälla på båda ställena, är alla utskrifter samlade i textfiler, vilka innehåller följande utskrifter:

MENY.DAT:

```
********************** DIGITAL REGLERING AV LINDA **********************

        MÖJLIGA REGLERMODER:
         0 = Tillfälligt avstängd datorstyrning (process igång)
         1 = Banspänningsreglering rulle 1/Hastighetsreglering rulle 2
         2 = Banspänningsreglering rulle 2/Hastighetsreglering rulle 1
         3 = Banspänning/Banspännings-reglering
         4 = Manuell momentstyrning
         5 = Manuellt moment rulle 1, Banspänningsreglering rulle 2
         6 = Banspänningsreglering rulle 1, Manuellt moment rulle 2
         7 = Manuellt moment rulle 1, Hastighetsreglering rulle 2
         8 = Hastighetsreglering rulle 1, Manuellt moment rulle 2
         9 = Styrning av börvärden Banspänning/Hastighets-reglering
        10 = Styrning av börvärden Banspänning/Banspännings-reglering
        99 = Stanna reglerprogrammet

        ALTERNATIV DISPLAY:
         N = Numeriskt presenterade mätvärden
        G1 = Grafisk presentation av värde 1
        G2 = Grafisk presentation av värde 2
         G = Grafisk presentation av bägge värdena
         H = FÖRKLARING AV MÖJLIGA OPERATÖRSKOMMANDON

VÄLJ REGLERMOD:
```

SIPCOM.DAT:

```
FÖLJANDE KOMMANDON ANVÄNDS FÖR STYRNING AV LINDA-PROCESSEN:
===========================================================


"BLANK" = Flytta cursorn till nästa plats där indata skall inmatas.
S,s     = Sänd värden på skärmen till reglerprocessen.
D,d     = Förnya displayen.
A,a     = Avsluta körningen av operatörskommunikationen, återvänd till menyn.

1-6     = Öka motsvarande siffra i aktuellt värde (cursorn) med 1.
          1 = första siffran = 1000-tal
          2 = andra siffran = 100-tal osv.
Q,q-Y,y = Minska motsvarande siffra i aktuellt tal.
          Q,q = första siffran = 1000-tal osv.

O,o     = Öppna fil för loggning av data.
K,k     = Stäng fil för loggning av data, ny fil kan öppnas med O.
L,l     = Logga tillsvidare.
P,p     = Gör en paus i loggning (återstart med L, eller stäng med K).


Tryck RETURN för att återvända till menyn !
 :
```

OPCAV.DAT:

```
        OPERATÖRSKOMMUNIKATIONEN ÄR AVSTÄNGD
```

FELMOD.DAT:

```
FELAKTIG REGLERMOD, FÖRSÖK IGEN !
```

FELDIS.DAT:

```
FELAKTIG DISPLAYKOD (N,G,G1,G2,H), FÖRSÖK IGEN !
```

Utskrift av MENY sker efter att alla ingående processer utom OPCOM, blivit startade med givna default-värden. Utskrift av SIPCOM sker vid kommandot H. När korrekta koder för reglermod och display erhållits startas OPCOM med angiven fil som indata. När OPCOM stannats med kommandot A, skrives OPCAV ut, varefter återhopp till utskrift av MENY göres utan att någon annan process stannas. Stopp av alla i systemet ingående processer och avslutning av hela programkörningen sker vid angivande av reglermod 99.

Listning av startfilerna:

START.COM:

```
$ ON CONTROL_Y THEN GOTO END
$ ON ERROR THEN GOTO END
$ SET NOON
$!
$ run A300,310AINIT
$ @A300,310ACLOCK.STV
$ @A300,311ALINDA.STV
$ @A300,311ALINSIM.STV
$!
$ WAIT 00:00:03
$ DEFINE/USER_MODE OPCSCR A300,311AOPCSCR.DAT
$!
$ MODVAL:
$ TYPE A300,311AMENY.DAT
$!
$ INQUIRE MODE "VÄLJ REGLERMOD"
$!
```

```
$ IF MODE .EQS. "0" THEN GOTO LBL00
$ IF MODE .EQS. "99" THEN GOTO LBL99
$!
$ DISVAL:
$ INQUIRE DISPLAY "VALJ DISPLAY"
$!
$ IF DISPLAY .EQS. "H" THEN GOTO SHHELP
$!
$ IF MODE .EQS. "1" THEN GOTO LBL01
$ IF MODE .EQS. "2" THEN GOTO LBL02
$ IF MODE .EQS. "3" THEN GOTO LBL03
$ IF MODE .EQS. "4" THEN GOTO LBL04
$ IF MODE .EQS. "5" THEN GOTO LBL05
$ IF MODE .EQS. "6" THEN GOTO LBL06
$ IF MODE .EQS. "7" THEN GOTO LBL07
$ IF MODE .EQS. "8" THEN GOTO LBL08
$ IF MODE .EQS. "9" THEN GOTO LBL09
$ IF MODE .EQS. "10" THEN GOTO LBL10
$!
$ TYPE A300,311AFELMOD.DAT
$ GOTO MODVAL
$!
$ SHHELP:
$ TYPE A300,311ASIPCOM.DAT
$ INQUIRE DUMMY " "
$ TYPE A300,311AMENY.DAT
$ GOTO DISVAL
$!
$!
$ LBL00:
$ COPY A300,311ASCRN0.DAT A300,311AOPCSCR.DAT
$ GOTO OPCRUN
$ LBL01:
$ IF DISPLAY .EQS. "N" THEN COPY A300,311ASCRN1.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G" THEN COPY A300,311ASCRG1.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G1" THEN COPY A300,311ASCR11.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G2" THEN COPY A300,311ASCR21.DAT A300,311AOPCSCR.DAT
$ GOTO DTEST
$ LBL02:
$ IF DISPLAY .EQS. "N" THEN COPY A300,311ASCRN2.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G" THEN COPY A300,311ASCRG2.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G1" THEN COPY A300,311ASCR12.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G2" THEN COPY A300,311ASCR22.DAT A300,311AOPCSCR.DAT
$ GOTO DTEST
$ LBL03:
$ IF DISPLAY .EQS. "N" THEN COPY A300,311ASCRN3.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G" THEN COPY A300,311ASCRG3.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G1" THEN COPY A300,311ASCR13.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G2" THEN COPY A300,311ASCR23.DAT A300,311AOPCSCR.DAT
$ GOTO DTEST
$ LBL04:
$ IF DISPLAY .EQS. "N" THEN COPY A300,311ASCRN4.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G" THEN COPY A300,311ASCRG4.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G1" THEN COPY A300,311ASCR14.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G2" THEN COPY A300,311ASCR24.DAT A300,311AOPCSCR.DAT
$ GOTO DTEST
$ LBL05:
$ IF DISPLAY .EQS. "N" THEN COPY A300,311ASCRN5.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G" THEN COPY A300,311ASCRG5.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G1" THEN COPY A300,311ASCR15.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G2" THEN COPY A300,311ASCR25.DAT A300,311AOPCSCR.DAT
$ GOTO DTEST
$ LBL06:
$ IF DISPLAY .EQS. "N" THEN COPY A300,311ASCRN6.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G" THEN COPY A300,311ASCRG6.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G1" THEN COPY A300,311ASCR16.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G2" THEN COPY A300,311ASCR26.DAT A300,311AOPCSCR.DAT
$ GOTO DTEST
$ LBL07:
$ IF DISPLAY .EQS. "N" THEN COPY A300,311ASCRN7.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G" THEN COPY A300,311ASCRG7.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G1" THEN COPY A300,311ASCR17.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G2" THEN COPY A300,311ASCR27.DAT A300,311AOPCSCR.DAT
$ GOTO DTEST
$ LBL08:
$ IF DISPLAY .EQS. "N" THEN COPY A300,311ASCRN8.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G" THEN COPY A300,311ASCRG8.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G1" THEN COPY A300,311ASCR18.DAT A300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G2" THEN COPY A300,311ASCR28.DAT A300,311AOPCSCR.DAT
$ GOTO DTEST
```

```
$ LBL09:
$ IF DISPLAY .EQS. "N" THEN COPY Ä300,311ASCRN9.DAT Ä300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G" THEN COPY Ä300,311ASCRG9.DAT Ä300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G1" THEN COPY Ä300,311ASCR19.DAT Ä300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G2" THEN COPY Ä300,311ASCR29.DAT Ä300,311AOPCSCR.DAT
$ GOTO DTEST
$ LBL10:
$ IF DISPLAY .EQS. "N" THEN COPY Ä300,311ASCRNA.DAT Ä300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G" THEN COPY Ä300,311ASCRGA.DAT Ä300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G1" THEN COPY Ä300,311ASCR1A.DAT Ä300,311AOPCSCR.DAT
$ IF DISPLAY .EQS. "G2" THEN COPY Ä300,311ASCR2A.DAT Ä300,311AOPCSCR.DAT
$ GOTO DTEST
$!
$!
$ DTEST:
$ IF DISPLAY .EQS. "N" THEN GOTO OPCRUN
$ IF DISPLAY .EQS. "G" THEN GOTO OPCRUN
$ IF DISPLAY .EQS. "G1" THEN GOTO OPCRUN
$ IF DISPLAY .EQS. "G2" THEN GOTO OPCRUN
$ TYPE Ä300,311AFELDIS.DAT
$ GOTO DISVAL
$!
$ OPCRUN:
$ PURGE Ä300,311AOPCSCR.DAT
$ RUN DM0:Ä300,310AOPCOM
$!
$ TYPE Ä300,311AOPCAV.DAT
$ GOTO MODVAL
$!
$ LBL99:
$ STOP "XDSKOUT"
$ STOP "XLINSIM"
$ STOP "XLINDA"
$ STOP "XCLOCK"
$!
$ END:
```

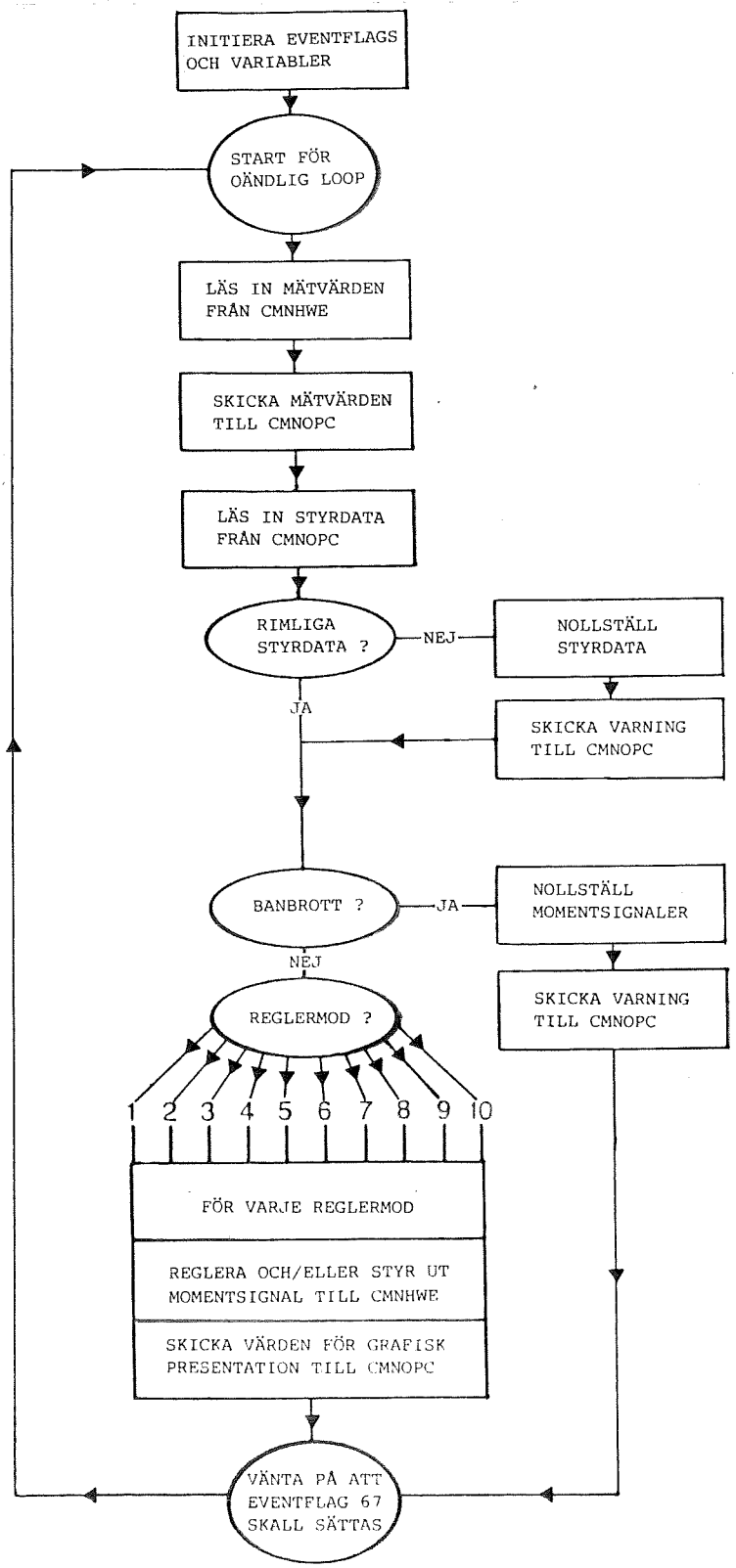
START.CMD:

```
; Denna fil är en översättning av START.COM från
; VAX-datorns till PDP-datorns kommandospråk.
; Större delen av filen ingår ej i dokumentationen
; då PDP-kommandot . missförstås av VAX-systemet.
;
set terminal /full/nowrap/form
run Ä300,310AINIT
@Ä300,310ACLOCK.STP
@Ä300,310ARTPHWE.STP
;@Ä300,310ADSKOUT.STP
;BORTTAGET PÅ GRUND AV PLATSBRIST I PDP
@Ä300,311ALINDA.STP
;
; %DOCEND%
;
.DISABLE DISPLAY
.ENABLE DECIMAL
;
.MODVAL:
TYPE Ä300,311AMENY.DAT
.ASKN MODE VÄLJ REGLERMOD >
;
.IF MODE = 0 .GOTO LBL00
.IF MODE = 99 .GOTO LBL99
;
.DISVAL:
.ASKS DISP VÄLJ DISPLAY >
;
.IF DISP = "H" .GOTO SHHELP
;
.IF MODE = 1 .GOTO LBL01
.IF MODE = 2 .GOTO LBL02
.IF MODE = 3 .GOTO LBL03
.IF MODE = 4 .GOTO LBL04
.IF MODE = 5 .GOTO LBL05
.IF MODE = 6 .GOTO LBL06
.IF MODE = 7 .GOTO LBL07
.IF MODE = 8 .GOTO LBL08
.IF MODE = 9 .GOTO LBL09
.IF MODE = 10 .GOTO LBL10
;
TYPE Ä300,311AFELMOD.DAT
.GOTO MODVAL
;
.SHHELP:
TYPE Ä300,311ASIPCOM.DAT
.ASKN DUMMY
```

```
TYPE A300,311AMENY.DAT
.GOTO DISVAL
;
.LBL00:
COPY A300,311ASCRN0.DAT A300,311AOPCSCR.DAT
.GOTO OPCRUN
.LBL01:
.IF DISP = "N" COPY A300,311ASCRN1.DAT A300,311AOPCSCR.DAT
.IF DISP = "G" COPY A300,311ASCRG1.DAT A300,311AOPCSCR.DAT
.IF DISP = "G1" COPY A300,311ASCR11.DAT A300,311AOPCSCR.DAT
.IF DISP = "G2" COPY A300,311ASCR21.DAT A300,311AOPCSCR.DAT
.GOTO DTEST
.LBL02:
.IF DISP = "N" COPY A300,311ASCRN2.DAT A300,311AOPCSCR.DAT
.IF DISP = "G" COPY A300,311ASCRG2.DAT A300,311AOPCSCR.DAT
.IF DISP = "G1" COPY A300,311ASCR12.DAT A300,311AOPCSCR.DAT
.IF DISP = "G2" COPY A300,311ASCR22.DAT A300,311AOPCSCR.DAT
.GOTO DTEST
.LBL03:
.IF DISP = "N" COPY A300,311ASCRN3.DAT A300,311AOPCSCR.DAT
.IF DISP = "G" COPY A300,311ASCRG3.DAT A300,311AOPCSCR.DAT
.IF DISP = "G1" COPY A300,311ASCR13.DAT A300,311AOPCSCR.DAT
.IF DISP = "G2" COPY A300,311ASCR23.DAT A300,311AOPCSCR.DAT
.GOTO DTEST
.LBL04:
.IF DISP = "N" COPY A300,311ASCRN4.DAT A300,311AOPCSCR.DAT
.IF DISP = "G" COPY A300,311ASCRG4.DAT A300,311AOPCSCR.DAT
.IF DISP = "G1" COPY A300,311ASCR14.DAT A300,311AOPCSCR.DAT
.IF DISP = "G2" COPY A300,311ASCR24.DAT A300,311AOPCSCR.DAT
.GOTO DTEST
.LBL05:
.IF DISP = "N" COPY A300,311ASCRN5.DAT A300,311AOPCSCR.DAT
.IF DISP = "G" COPY A300,311ASCRG5.DAT A300,311AOPCSCR.DAT
.IF DISP = "G1" COPY A300,311ASCR15.DAT A300,311AOPCSCR.DAT
.IF DISP = "G2" COPY A300,311ASCR25.DAT A300,311AOPCSCR.DAT
.GOTO DTEST
.LBL06:
.IF DISP = "N" COPY A300,311ASCRN6.DAT A300,311AOPCSCR.DAT
.IF DISP = "G" COPY A300,311ASCRG6.DAT A300,311AOPCSCR.DAT
.IF DISP = "G1" COPY A300,311ASCR16.DAT A300,311AOPCSCR.DAT
.IF DISP = "G2" COPY A300,311ASCR26.DAT A300,311AOPCSCR.DAT
.GOTO DTEST
.LBL07:
.IF DISP = "N" COPY A300,311ASCRN7.DAT A300,311AOPCSCR.DAT
.IF DISP = "G" COPY A300,311ASCRG7.DAT A300,311AOPCSCR.DAT
.IF DISP = "G1" COPY A300,311ASCR17.DAT A300,311AOPCSCR.DAT
.IF DISP = "G2" COPY A300,311ASCR27.DAT A300,311AOPCSCR.DAT
.GOTO DTEST
.LBL08:
.IF DISP = "N" COPY A300,311ASCRN8.DAT A300,311AOPCSCR.DAT
.IF DISP = "G" COPY A300,311ASCRG8.DAT A300,311AOPCSCR.DAT
.IF DISP = "G1" COPY A300,311ASCR18.DAT A300,311AOPCSCR.DAT
.IF DISP = "G2" COPY A300,311ASCR28.DAT A300,311AOPCSCR.DAT
.GOTO DTEST
.LBL09:
.IF DISP = "N" COPY A300,311ASCRN9.DAT A300,311AOPCSCR.DAT
.IF DISP = "G" COPY A300,311ASCRG9.DAT A300,311AOPCSCR.DAT
.IF DISP = "G1" COPY A300,311ASCR19.DAT A300,311AOPCSCR.DAT
.IF DISP = "G2" COPY A300,311ASCR29.DAT A300,311AOPCSCR.DAT
.GOTO DTEST
.LBL10:
.IF DISP = "N" COPY A300,311ASCRNA.DAT A300,311AOPCSCR.DAT
.IF DISP = "G" COPY A300,311ASCRGA.DAT A300,311AOPCSCR.DAT
.IF DISP = "G1" COPY A300,311ASCR1A.DAT A300,311AOPCSCR.DAT
.IF DISP = "G2" COPY A300,311ASCR2A.DAT A300,311AOPCSCR.DAT
.GOTO DTEST
;
;
.DTEST:
.IF DISP = "N" .GOTO OPCRUN
.IF DISP = "G" .GOTO OPCRUN
.IF DISP = "G1" .GOTO OPCRUN
.IF DISP = "G2" .GOTO OPCRUN
TYPE A300,311AFELDIS.DAT
.GOTO DISVAL
;
.OPCRUN:
PURGE A300,311AOPCSCR.DAT
run A300,310AOPCOM
;
TYPE A300,311AOPCAV.DAT
.GOTO MODVAL
;
.LBL99:
;
.ENABLE DISPLAY
```

8.8 Applikationsprogrammet

Vad applikationsprogrammet skall utföra kan enklast sammanfattas i ett flödesschema (Figur 8.9).



Figur 8.9 Flödesschema för processen Linda.

Själva regelringen sker med underprogrammet PIDREG, utstyrning av utdata med underprogrammet STYRUT och omvandling från real till integer med under programmet RIOMV.
I regleralgoritmen för PIDREG skall vissa värden sparas mellan samplingarna. Dessa samlas i GLBLIN.INC som ser ut på följande sätt:

```
c       DETTA ÄR GLOBALA VARIABLER FÖR APPLIKATIONEN LINDA.

c       Gamla värden som skall bli ihågkommna:
        real*4  GOLD(1:4),GOLDER(1:4),GMOM(1:2)
        integer*2 GMOD
c       GOLD,GOLDER är värden från tidigare samplingar
c       i regleralgoritm.
c       GMOM är föregående samplings momentsignal.
c       GMOD är föregående samplings inställda reglermod.


        common /GLBLIN/
     1
     1        GOLD,GOLDER,
     1        GMOM,
     1        GMOD
```

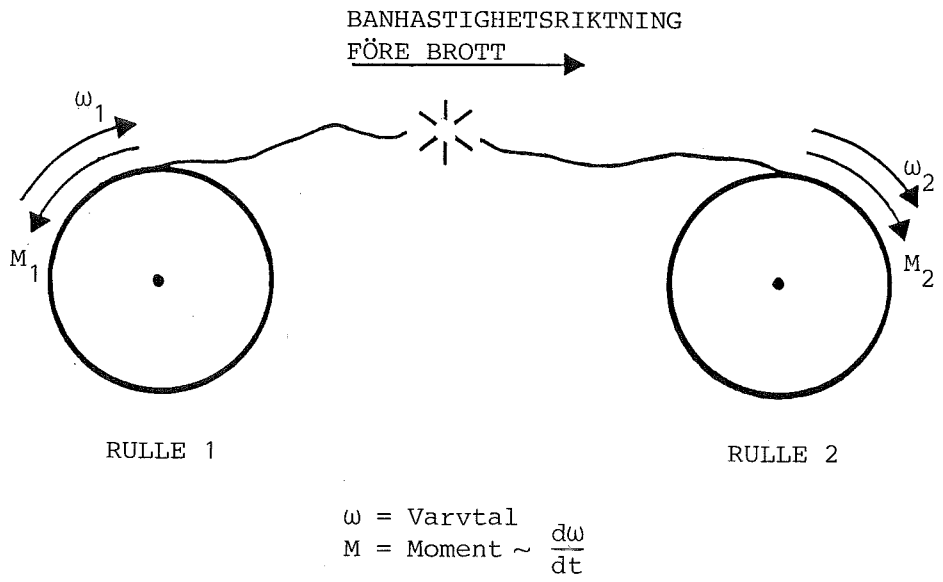
För att få konsekvens i uträkningar omvandlas alla värden till volt. Följande omräkningsfaktorer kommer då att gälla (se kapitel 7.5).

- Analoga värden från CMNHWE: dividera med 200,5
- Analoga värden till CMNHWE: multiplicera med 200,5
- Numeriska värden från CMNOPC: ges i volt
- Numeriska värden till CMNOPC: ges i volt
- Grafiska värden till CMNOPC: multiplicera med 100

Banbrottshanteringen kan ske på olika sätt. Det analoga reglersystemet för LINDA sätter börvärdena för regulatorerna till 0

vid indikerad banbrottssignal. Detta får effekten att regulatorerna strävar efter att bromsa rullarna med drivmomentet. Detta fungerar bara på den bromsande rullen, rulle 1 i Figur 8.10, som redan i banbrottsögonblicket har ett bromsande moment och därigenom inte får någon kraftig rotation.

Banbrottsproblemet blir alltså störst på den drivande rullen (rulle 2 i Figur 8.10), som i banbrottsögonblicket får en vinkelacceleration i samma riktning som den rådande rotationen. För att bromsa denna skulle det krävas en negativ momentsignal, vilket inte medgives av den befintliga utrustningen. Att koppla in någon form av reglering på denna problemrulle medför endast att momentsignalen blir noll. Med hänsyn tagen till rådande omständigheter, räcker det att sätta båda momentsignalerna till noll vid indikering av banbrott, vilket är utfört i denna reglerprocess.



Figur 8.10 Kraftverkan på rullarna vid banbrott.

Enligt mallen från kapitel 7 blir källkoden till LINDA:

```
c IDENT A300,311ALINDA.FOR

      program LINDA


c*****************************************************************
c
c       REGULATOR FÖR DIREKT DIGITAL REGLERING AV LINDA
c       Denna version reglerar med en PID-regulator, där
c       operatören bestämmer konstanterna.
c
c*****************************************************************

c---------------------------------------------------------------

c       TABELL ÖVER INDATA FRÅN CMNHWE (PROCESSEN):

c       Ofiltrerade insignaler:
c       I3643(4)=LTEMP(1)=Banhastighet, börvärde
c       I3643(5)=LTEMP(2)=Banspänning rulle 1, börvärde
c       I3643(6)=LTEMP(3)=Banspänning rulle 2, börvärde
c       I3643(7)=LBROTT  =Banbrottsindikering

c       Insignaler med lågpassfilter, 3 dB gräns = 11 Hz
c       I3644(0)=LTEMP(4)=Banhastighet, ärvärde
c       I3644(1)=LTEMP(5)=Banspänning rulle 1, ärvärde
c       I3644(2)=LTEMP(6)=Banspänning rulle 2, ärvärde
c       I3644(3)=LTEMP(7)=Moment rulle 1,
c       I3644(4)=LTEMP(8)=Moment rulle 2,
c       I3644(5)=LTEMP(9)=Varvtal rulle 1
c       I3644(6)=LTEMP(10)=Varvtal rulle 2

c       TABELL ÖVER UTDATA TILL CMNHWE (PROCESSEN):

c       03820(0)=Digital mask som anger inställd reglermod,
c                bit 15 - extern momentstyrning
c                bit 14 - extern börvärdesstyrning av
c                banspänning rulle 1
c                bit 13 - extern börvärdesstyrning av
c                banspänning rulle 2
c                bit 12 - extern börvärdesstyrning av
c                banhastighet
c                bit 15 är minst signifikant, komplement
c                till önskad utsignal skall anges.
c       05531(1)=Utdata vars betydelse beror på reglermod:
c                Extern börvärdesstyrning, banspänning R 1
c                Extern momentstyrning, rulle 1
c       05531(2)=Utdata vars betydelse beror på reglermod:
c                Extern börvärdesstyrning, banspänning R 2
c                Extern momentstyrning, rulle 2
c                Extern börvärdesstyrning av banhastighet

c       TABELL ÖVER UTDATA TILL CMNOPC (OPCOM):

c       OPCIUT(1)=Banbrottsindikering
c       OPCIUT(2)=Indikering av mättad momentsignal rulle 1
c       OPCIUT(3)=Indikering av mättad momentsignal rulle 2
c       OPCIUT(4)=Indikering av orimliga styrdata
```

```
c       OPCRUT(1)=Banhastighet, börvärde
c       OPCRUT(2)=Banspänning rulle 1, börvärde
c       OPCRUT(3)=Banspänning rulle 2, börvärde
c       OPCRUT(4)=Banhastighet, ärvärde
c       OPCRUT(5)=Banspänning rulle 1, ärvärde
c       OPCRUT(6)=Banspänning rulle 2, ärvärde
c       OPCRUT(7)=Moment, rulle 1
c       OPCRUT(8)=Moment, rulle 2
c       OPCRUT(9)=Varvtal, rulle 1
c       OPCRUT(10)=Varvtal, rulle 2

c       TABELL ÖVER INDATA FRAN CMNOPC (OPCOM):

c       OPCRIN(1)=Inställd reglermod
c       OPCRIN(2)=LBVIN(1)=Indata vars betydelse beror på
c                         inställd reglermod
c                         Börvärde, banspänning rulle 1
c                         eller rulle 2
c                         Momentstyrning, rulle 1
c       OPCRIN(3)=LBVIN(2)=Indata vars betydelse beror på
c                         inställd reglermod
c                         Börvärde, banspänning rulle 2
c                         Momentstyrning, rulle 2
c                         Börvärde, banhastighet

c       OPCRIN(4)=Reglerkonst. P-del, banspänningsreglering
c       OPCRIN(5)=Reglerkonst. P-del, hastighetsreglering
c       OPCRIN(6)=Reglerkonst. I-del, banspänningsreglering
c       OPCRIN(7)=Reglerkonst. I-del, hastighetsreglering
c       OPCRIN(8)=Reglerkonst. D-del, banspänningsreglering
c       OPCRIN(9)=Reglerkonst. D-del, hastighetsreglering

c-------------------------------------------------------------

c       %DOCNOD% LBR $SIP$:LBRXXX.FOR

c       %DOCNOD% CON $SIP$:CMNHWE.INC
        include 'A300,310ACMNHWE.INC'

c       %DOCNOD% CON $SIP$:CMNOPC.INC
        include 'A300,310ACMNOPC.INC'

c       %DOCNOD% GLB $LINDA$:GLBLIN.INC
        include 'A300,311AGLBLIN.INC'

c       %DOCNOD% PRI $SIP$:DUMMY.DUM PRI=25

c       Filer som medtages i dokumentationen:
c       %DOCNOD% DOC $LINDA$:GLBLIN.INC Globala variabler
c       %DOCNOD% DOC $LINDA$:START.COM Startfil i VAX
c       %DOCNOD% DOC $LINDA$:START.CMD Startfil i PDP

c       %DOCEND%

        integer*2 LSTAT,LCOUNT,LDUMMY,LMOD
c       LSTAT är statusflagga vid anrop av underprogram
c       LCOUNT är en räknare
c       LDUMMY allmän dummyvariabel
c       LMOD anger aktuell reglermod
```

```
      real*4 LTEMP(1:OPCRUA),LINMAX,LSAMP,LBVIN(1:2)
      real*4 LBROTT,LALARM
c     LTEMP är mätvärden omvandlade till enheten volt
c     LINMAX är takvärde för rimliga indata
c     LSAMP är gällande samplingsintervall
c     LBVIN är av operatören givet börvärde
c     LBROTT är analog banbrottssignal i volt
c     LALARM är nivån i volt då alarm för banbrott ges

c-----------------------------------------------------------------

      print *,'LINDAs digitala regulator har startat'

      call XCRGEF(LSTAT)

c     Stäng av datorstyrning under initieringsskedet:
      03820(0)=-1
c     Digital utsignal är "bakvänd", logisk etta = 0 V

c     Sätt alla variabler till sina startvärden:

      LINMAX=15.0
      LALARM=-5.0

      GMOM(1)=I3644(3)
      GMOM(2)=I3644(4)

      do 11 LCOUNT=1,4
          GOLD(LCOUNT)=0
          GOLDER(LCOUNT)=0
          OPCGUT(LCOUNT)=0
11    continue

c------- start LOOP ------------------------------------------------

9999  continue

c     Samplingstiden blir INSTALLD TID*LDELTA*20 ms,
c     LDELTA finns i processen CLOCK, LDELTA=2
      LSAMP=HWEWTM(67)*40.0

c     Läs in alla mätvärden.
c     Konstanterna ger mätvärden i volt:

      LBROTT=(real(10*I3643(7)))/2005

      do 22 LCOUNT=4,10
          LTEMP(LCOUNT)=(real(10*I3644(LCOUNT-4)))/2005
          OPCRUT(LCOUNT)=LTEMP(LCOUNT)
22    continue

      do 33 LCOUNT=1,3
          LTEMP(LCOUNT)=(real(10*I3643(LCOUNT+3)))/2005
          OPCRUT(LCOUNT)=LTEMP(LCOUNT)
33    continue

c     Läs in börvärden eller manuella styrsignaler:
      LBVIN(1)=OPCRIN(2)
      LBVIN(2)=OPCRIN(3)
```

```
c       Kontrollera att operatören angett rimliga värden
        LDUMMY=0
        do 44 LCOUNT=1,2
            if (abs(LBVIN(LCOUNT)).gt.LINMAX) then
                LDUMMY=1
                LBVIN(LCOUNT)=0
            endif
44      continue
        OPCIUT(4)=LDUMMY

c       Kontrollera om banbrott har inträffat:
        if (LBROTT.lt.LALARM) then
            LMOD=11
            OPCIUT(1)=1
        else
            call RIOMV(LMOD,OPCRIN(1),LDUMMY)
            OPCIUT(1)=0
        endif

c       Om reglermod har ändrats, nollställ gamla
c       värden (ej momentsignal):
        if (LMOD.ne.GMOD) then
            do 55 LCOUNT=1,4
                GOLD(LCOUNT)=0
                GOLDER(LCOUNT)=0
55          continue
        endif
        GMOD=LMOD

c-------- Behandla de olika fallen av reglermoder: ----------

        goto(1010,1020,1030,1040,1050,1060,1070,1080,1090,
        1     1100,1111) LMOD

c           LMOD=0 (eller LMOD<0 eller LMOD>11, dvs. fel)
c           => avstängd datorstyrning:
            03820(0)=-1
            goto 1999

c           LMOD=1 => banspänningreglering rulle 1
c                     /hastghetsreglering rulle 2:
1010        03820(0)=-2
            call PIDREG
        1   (.false.,LBVIN(1),LTEMP(5),05531(1),OPCRIN(4),
        1   OPCRIN(6),OPCRIN(8),LSAMP,OPCIUT(2),1)
            call PIDREG (.true.,LBVIN(2),LTEMP(4),05531(2),
        1   OPCRIN(5),OPCRIN(7),OPCRIN(9),LSAMP,OPCIUT(3),2)
c           Rita med enheten 100volt:
            call RIOMV(OPCGUT(3),LTEMP(5)*100,LDUMMY)
            call RIOMV(OPCGUT(4),LTEMP(4)*100,LDUMMY)
            goto 1999

c           LMOD=2 => hastghetsreglering rulle 1
c                     /banspänningreglering rulle 2:
1020        03820(0)=-2
            call PIDREG
        1   (.false.,LBVIN(1),LTEMP(6),05531(2),OPCRIN(4),
        1   OPCRIN(6),OPCRIN(8),LSAMP,OPCIUT(3),2)
            call PIDREG (.true.,-LBVIN(2),-LTEMP(4),05531(1),
        1   OPCRIN(5),OPCRIN(7),OPCRIN(9),LSAMP,OPCIUT(2),1)
```

```
c               Rita med enheten 100volt:
                call RIOMV(OPCGUT(3),LTEMP(6)*100,LDUMMY)
                call RIOMV(OPCGUT(4),LTEMP(4)*100,LDUMMY)
                goto 1999

c               LMOD=3: => banspänning/banspänningsreglering:
1030            03820(0)=-2
                call PIDREG
           1    (.false.,LBVIN(1),LTEMP(5),05531(1),OPCRIN(4),
           1    OPCRIN(6),OPCRIN(8),LSAMP,OPCIUT(2),1)
                call PIDREG
           1    (.false.,LBVIN(2),LTEMP(6),05531(2),OPCRIN(4),
           1    OPCRIN(6),OPCRIN(8),LSAMP,OPCIUT(3),2)
                call RIOMV(OPCGUT(3),LTEMP(5)*100,LDUMMY)
                call RIOMV(OPCGUT(4),LTEMP(6)*100,LDUMMY)
                goto 1999

c               LMOD=4 => manuell momentstyrning:
1040            03820(0)=-2
                call STYRUT(LBVIN(1),05531(1),2047,0,OPCIUT(2))
                call STYRUT(LBVIN(2),05531(2),2047,0,OPCIUT(3))
                call RIOMV(OPCGUT(3),LTEMP(7)*100,LDUMMY)
                call RIOMV(OPCGUT(4),LTEMP(8)*100,LDUMMY)
                goto 1999

c               LMOD=5 => manuellt moment rulle 1
c                       / banspänning rulle 2:
1050            03820(0)=-2
                call STYRUT(LBVIN(1),05531(1),2047,0,OPCIUT(2))
                call PIDREG
           1    (.false.,LBVIN(2),LTEMP(6),05531(2),OPCRIN(4),
           1    OPCRIN(6),OPCRIN(8),LSAMP,OPCIUT(3),2)
                call RIOMV(OPCGUT(3),LTEMP(7)*100,LDUMMY)
                call RIOMV(OPCGUT(4),LTEMP(6)*100,LDUMMY)
                goto 1999

c               LMOD=6 => banspänning rulle 1
c                       / manuellt moment rulle 2:
1060            03820(0)=-2
                call PIDREG
           1    (.false.,LBVIN(1),LTEMP(5),05531(1),OPCRIN(4),
           1    OPCRIN(6),OPCRIN(8),LSAMP,OPCIUT(2),1)
                call STYRUT(LBVIN(2),05531(2),2047,0,OPCIUT(3))
                call RIOMV(OPCGUT(3),LTEMP(5)*100,LDUMMY)
                call RIOMV(OPCGUT(4),LTEMP(8)*100,LDUMMY)
                goto 1999

c               LMOD=7 => manuellt moment rulle 1
c                       / hastighet rulle 2:
1070            03820(0)=-2
                call STYRUT(LBVIN(1),05531(1),2047,0,OPCIUT(2))
                call PIDREG (.true.,LBVIN(2),LTEMP(4),05531(2),
           1    OPCRIN(5),OPCRIN(7),OPCRIN(9),LSAMP,OPCIUT(3),2)
                call RIOMV(OPCGUT(3),LTEMP(7)*100,LDUMMY)
                call RIOMV(OPCGUT(4),LTEMP(4)*100,LDUMMY)
                goto 1999

c               LMOD=8 => hastighet rulle 1
c                       / manuellt moment rulle 2:
```

```
1080          03820(0)=-2
              call PIDREG (.true.,-LBVIN(1),-LTEMP(4),05531(1),
             1  OPCRIN(5),OPCRIN(7),OPCRIN(9),LSAMP,OPCIUT(2),1)
              call STYRUT(LBVIN(2),05531(2),2047,0,OPCIUT(3))
              call RIOMV(OPCGUT(3),LTEMP(4)*100,LDUMMY)
              call RIOMV(OPCGUT(4),LTEMP(8)*100,LDUMMY)
              goto 1999

c             LMOD=9 => börvärdesstyrning av
c                       banspänning/hastighet:
1090          03820(0)=-13
              call STYRUT(LBVIN(1),05531(1),2047,0,LDUMMY)
              call STYRUT(LBVIN(2),05531(2),2047,-2047,LDUMMY)
              call RIOMV(OPCGUT(3),LTEMP(4)*100,LDUMMY)
              call RIOMV(OPCGUT(4),LTEMP(5)*100,LDUMMY)
              goto 1999

c             LMOD=10 => börvärdesstyrning av
c                        banspänning/banspänning:
1100          03820(0)=-7
              call STYRUT(LBVIN(1),05531(1),2047,0,LDUMMY)
              call STYRUT(LBVIN(2),05531(2),2047,0,LDUMMY)
              call RIOMV(OPCGUT(3),LTEMP(5)*100,LDUMMY)
              call RIOMV(OPCGUT(4),LTEMP(6)*100,LDUMMY)
              goto 1999

c             LMOD=11 => banbrott! Sätt momenten till 0:
1111          03820(0)=-2
              call STYRUT(0,05531(1),2047,0,OPCIUT(2))
              call STYRUT(0,05531(2),2047,0,OPCIUT(3))
              call RIOMV(OPCGUT(3),LTEMP(7)*100,LDUMMY)
              call RIOMV(OPCGUT(4),LTEMP(8)*100,LDUMMY)
              goto 1999

1999      continue

          call RIOMV(OPCGUT(1),LBVIN(1)*100,LDUMMY)
          call RIOMV(OPCGUT(2),LBVIN(2)*100,LDUMMY)

c-------- end Behandling av olika moder ----------------------

          call     XCLREF(67,LSTAT)
          call     XWEF(67,LSTAT)

          goto 9999

c-------- end LOOP -------------------------------------------

          end

c======== end LINDA ==========================================

          subroutine PIDREG(PHAST,PREFIN,PVALIN,PVALUT,PPPAR,
          1                 PIPAR,PDPAR,PSAMP,PSTAT,PRULLE)


c         Denna rutin utför själva regleringen med en
c         PID-regulator där PPPAR,PIPAR,PDPAR är P,I
c         resp. D-parametrarna givna av operatören

c         %DOCNOD% GLB $LINDA$:GLBLIN.INC
          include 'A300,311AGLBLIN.INC'
```

```
      logical PHAST
      real*4 PREFIN,PVALIN,PPPAR,PIPAR,PDPAR,PSAMP
      integer*2 PVALUT,PSTAT,PRULLE,LHAST
      real*4 LT0,LT1,LS0,LS1,LS2
c     PHAST   = True för hastighetsreglering, false annars
c     LHAST   = 2 för hastighetsreglering, 0 annars
c     PREFIN = Referenssignal från operatör
c     PVALIN = Insignal från processen
c     PVALUT = Styrsignal till processen
c     LT0... = Konstanter i regleralgoritm
c     PSTAT   = Kontroll om momentsignalen bottnar
c     PSAMP   = Samplingsintervall
c     PRULLE = Talar om vilken rulle man reglerar på

c-----------------------------------------------------------------------

c     Bestäm parametrar i polynom av bakåtskiftes
c     -operatorn: (Proportionalitetskonstanten tillkommer
c     i regleralgoritmen)
      if (PDPAR.le.0) PDPAR=0
      if (PIPAR.le.0) then
          LT0=1
          LS0=1+PDPAR/PSAMP
      else
          LT0=1+PSAMP/PIPAR
          LS0=1+PSAMP/PIPAR+PDPAR/PSAMP
      endif
      LT1=-1
      LS1=-1-2*PDPAR/PSAMP
      LS2=PDPAR/PSAMP

c     Kontrollera om hastighetsreglering:
c     (olika systemglobala vektorelement)
      if (PHAST) then
          LHAST=2
      else
          LHAST=0
      endif

c     Regleralgoritm:
      GMOM(PRULLE)=GMOM(PRULLE)+
     1(PREFIN*LT0-PVALIN*LS0-GOLD(PRULLE+LHAST))*PPPAR

c     Styrut erhållet värde:
      call STYRUT(GMOM(PRULLE),PVALUT,2047,0,PSTAT)

c     Anti-integratormättning:
      GMOM(PRULLE)=(real(10*PVALUT))/2005

c     Uppdatera variabler i "minnet":
      GOLD(PRULLE+LHAST)=-LT1*PREFIN+LS1*PVALIN+
     1LS2*GOLDER(PRULLE+LHAST)
      GOLDER(PRULLE+LHAST)=PVALIN

      return
      end

c======== end PIDREG ==========================================
```

```
      subroutine STYRUT(PREAL,PVALUT,PMAX,PMIN,PSTAT)

c     Styr ut värdet av PREAL till PVALUT och omvandlar
c     det från reellt värde i volt till digitalt
c     ord till D/A-omvandlare

      real*4     PREAL
      integer*2 PVALUT,PMAX,PMIN,PSTAT,LTEMP,LSTAT
c     PMAX   = Största tillåtna utsignal
c     PMIN   = Minsta tillåtna utsignal
c     PSTAT = 1 Anger att styrsignalen är mättad
c     LTEMP,LSTAT = Lokala variabler för omvanling

c-----------------------------------------------------------------------

      call RIOMV(LTEMP,PREAL*200.5,LSTAT)
      if ((LTEMP.ge.PMAX).or.(LSTAT.eq.1)) then
          PSTAT=1
          PVALUT=PMAX
      elseif ((LTEMP.le.PMIN).or.(LSTAT.eq.-1)) then
          PSTAT=1
          PVALUT=PMIN
      else
          PSTAT=0
          PVALUT=LTEMP
      endif

      return
      end

c======= end STYRUT ===================================================


      subroutine RIOMV(PINT,PREAL,PSTAT)

c     Denna rutin ger avrundning av det rella talet
c     PREAL till närmaste heltal, PINT. Vid integer
c     overflow fås värdet 0. PSTAT får värdet 1=för stort
c     tal, -1=för litet, 0=lagom.

      real*4 PREAL
      integer*2 PSTAT,PINT

c-----------------------------------------------------------------------

      if (PREAL.ge.32767.0) then
          PINT=0
          PSTAT=1
      elseif (PREAL.le.-32768.0) then
          PINT=0
          PSTAT=-1
      else
          PINT=nint(PREAL)
          PSTAT=0
      endif

      return
      end

c======= end RIOMV ====================================================
```

8.9 Simuleringsprogram

I enlighet med avsnitt 3.1 har en simuleringsmodul skapats som ger värden på indata till LINDA-programmet. Under utvecklingsarbetet har olika varianter av signalgenerering utnyttjats för att kunna testa olika delar och versioner av LINDA samt för att ge en åskådlig bild av hur SIP-systemet fungerar. Här redovisas ett exempel på hur simuleringsprogrammet LINSIM.FOR kan se ut:

```
c IDENT A300,311ALINSIM.FOR

      program LINSIM

c*************************************************************
c
c       Program för att simulera processen
c       LINDA på VAX-datorn
c
c*************************************************************

c       %DOCNOD% LBR $SIP$:LBRXXX.FOR

c       %DOCNOD% CON $SIP$:CMNHWE.INC
        include 'A300,310ACMNHWE.INC'

c       %DOCNOD% PRI $SIP$:DUMMY.DUM PRI=27

c       %DOCEND%

        integer*2 LSAMP,LSTAT,LMOD
c       LSAMP = Räknar antalet cykler som
c               LINSIM genomlöper
c       LSTAT = Statusflagga vid rutinanrop
c       LMOD  = Markerar av LINDA-programmet
c               inställd reglermod

c-------------------------------------------------------------

        print *,'Simuleringen av LINDA har börjat'

        call XCRGEF(LSTAT)

        LSAMP=0

c------- start LOOP ----------------------------------------

1000    continue

        LSAMP=LSAMP+1
        if (LSAMP.eq.1001) LSAMP=1
```

```
c       Simulering av banbrott:
        if (LSAMP.lt.20) then
            I3643(7)=-2000
        else
            I3643(7)=0
        endif


        LMOD=03820(0)+1
        if (mod(LMOD,2).eq.-1) then
            I3644(3)=05531(1)
c           Moment rulle 1
            I3644(4)=05531(2)
c           Moment rulle 2
         else
            I3644(3)=0
            I3644(4)=0
        endif

        LMOD=LMOD/2
        if (mod(LMOD,2).eq.-1) then
            I3643(6)=05531(2)
c           Banspänning rulle 2, börvärde
        else
            I3643(6)=300
        endif

        LMOD=LMOD/2
        if (mod(LMOD,2).eq.-1) then
            I3643(5)=05531(1)
c           Banspänning rulle 1, börvärde
        else
            I3643(5)=600
        endif

        LMOD=LMOD/2
        if (mod(LMOD,2).eq.-1) then
            I3643(4)=05531(2)
c           Hastighet, börvärde
        else
            I3643(4)=1100
        endif

        if (mod(LSAMP,10).gt.4) then
            I3644(0)=05531(2)+125-25*mod(LSAMP,10)
        else
            I3644(0)=05531(2)+25*mod(LSAMP,10)
        endif
c       Hastighet, ärvärde

        I3644(1)=05531(1)
c       Banspänning rulle 1, ärvärde

        I3644(2)=05531(2)+(mod(LSAMP,3))*2
c       Banspänning rulle 2, ärvärde
```

```
            I3644(5)=I3644(0)/50+LSAMP/100
c           Varvtal rulle 1, ärvärde

            I3644(6)=I3644(0)/50+abs(100-LSAMP)
c           Varvtal rulle 2, ärvärde

            call     XCLREF(69,LSTAT)
            call     XWEF(69,LSTAT)

            goto 1000

c-----------end LOOP-----------------------------------------------------

            end

c-----------end LINSIM---------------------------------------------------
```

REFERENSER

1. Ekman, Torgil och Eriksson, Göran:
Programmering i Fortran 77.
Studentlitteratur, Lund 1981.

2. Eriksson, Leif:
Försöksanläggning för studier av processer med snabblöpande pappersbanor. Kommande rapport från Tidningspappersbrukens Forskningslaboratorium (TFL), Djursholm.

3. Elmqvist, Hilding, Mattsson, Sven Erik och Olsson, Gustaf:
Datorer i reglersystem. Realtidsprogrammering. Institutionen för Reglerteknik. Tekniska Högskolan i Lund, Lund 1981.

4. Åström, Karl J. och Wittenmark, Björn:
Computer Controlled Systems. Theory and Design.
Prentic Hall INC, Englewood Cliffs, N.J. 1984.

5. Åström, Karl Johan:
Reglerteori.
Almqvist & Wiksell, Uppsala 1976.

Manualer:

6. Wallin, Anders:
Linda. Omrullningsmaskin för papper.
Del 1 - Allmän beskrivning och handledning. Tidnings-
pappersbrukens Forskningslaboratorium (TFL), 1984.

7. Wallin, Anders:
Linda. Omrullningsmaskin för papper.
Del II - Teknisk beskrivning och ritningar.
Tidningspappersbrukens Forskningslaboratorium T(FL), 1982.

8. OmniSystem. Manualer för RTP7432, RTP7435/82, RTP7437/37,
RTP8435/83, RTP7438/20, RTP7435/33, RTP7435/34, RTÅ7435/51,
RTP7436, RTP7455/30 samt RTP7455/20.
Computer Products INC, Fort Lauderdale 1981-1983.

9. DMØ:<300,310>Sip.Liz.
Dokumentationsfil, SIP-systemet.
Tidningspappersbrukens Forskningslaboratorium,
kontinuerlig uppdatering.

10. 4105 Computer Display Terminal
Operators Manual.
Textronix Inc. Beaverton, Oregon 1983.

11. 4105 Computer Display Terminal.
Programmers Reference.
Textronix Inc. Beaverton, Oregon 1983.

12. 4105 Computer Display Terminal.
Reference Guide.
Textronix Inc. Beaverton, Oregon 1983.

13. RSX-11M- Version 4.0. Real-Time Operating System.
Digital Equipment Corption, Maynard Massachusetts 1982.

14. PDP-11, Fortran 77/RSX, version 4.1.
Digital Equipment Corporation, Maynard, Massachusetts 1982.

15. VAX/VMX, Operating system, version 3.4.
Digital Equipment Corporation, Maynard, Massachusetts 1983.

16. VAX-11 Fortran, version 3.4.
Digital Equipment Corporation, Maynard, Massachusetts 1983.